ELSEVIER

Artificial Intelligence 116 (2000) 17–66

Artificial Intelligence

www.elsevier.com/locate/artint

Proving theorems by reuse

Christoph Walther *, Thomas Kolbe [1]

Fachbereich Informatik, Technische Universität Darmstadt, Alexanderstr. 10, D-64283 Darmstadt, Germany

Received 22 October 1997; received in revised form 7 June 1999

Abstract

We investigate the improvement of theorem proving by reusing previously computed proofs. We have developed and implemented the PLAGIATOR system which proves theorems by mathematical induction with the aid of a human advisor: If a base or step formula is submitted to the system, it tries to reuse a proof of a previously verified formula. If successful, labour is saved, because the number of required user interactions is decreased. Otherwise the human advisor is called for providing a hand crafted proof for such a formula, which subsequently—after some (automated) preparation steps—is stored in the system's memory, to be in stock for future reasoning problems. Besides the potential savings of resources, the performance of the overall system is improved, because necessary lemmata might be speculated as the result of an attempt to reuse a proof. The success of the approach is based on our techniques for *preparing* given proofs as well as by our methods for *retrieval* and *adaptation* of reuse candidates which are promising for future proof reuses. We prove the soundness of our approach and illustrate its performance with several examples. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Deduction and theorem proving; Machine learning; Problem solving and search; Knowledge representation; Analogy; Abstraction; Reuse

1. Introduction

The improvement of problem solvers by reusing previously computed solutions is an active research area of Artificial Intelligence, emerging in the methodologies of *explanation-based learning* (EBL) [22,28,65] and *analogical reasoning* (AR) [14,39, 68]. In EBL a problem's solution is analyzed, yielding an explanation why the solution succeeds. After generalization, the explanation is used for solving (similar) new problems.

* Corresponding author. Email: chr.walther@informatik.tu-darmstadt.de. This work was supported under grants no. Wa652/4-1,2,3 by the Deutsche Forschungsgemeinschaft as part of the focus program "Deduktion".

[1] Email: kolbe@informatik.tu-darmstadt.de.

0004-3702/00/$ – see front matter
PII: S0004-3702(99)00096-X

In AR a problem's solution guides the solution of (similar) new problems by suggesting corresponding inference steps. We present an approach for reusing *proofs* that combines ideas of EBL and AR as well as ideas from *abstraction* techniques [36,69].

We investigate the reuse of first-order proofs within the domain of automated mathematical induction [8,11,44,46,82] where similar conjectures often have similar proofs: [2] An induction theorem prover either proves a conjecture by first-order inferences or otherwise associates the conjecture with a finite set of *induction formulas* whose truth entail the truth of the conjecture (by means of some *induction axiom*). An *induction formula* $IH \rightarrow IC$ is either a *step* formula or a *base* formula in which case *IH* equals TRUE. Induction formulas form new conjectures serving as input to the prover, and the original conjecture is proved if eventually only (first-order) provable induction formulas, i.e., *valid* formulas, remain. Such formulas are proved by modifying the *induction conclusion IC* using axioms and the *induction hypothesis IH* until TRUE is inferred. Despite this regularity the search problem of deciding *when* and *where* to apply *which* axiom such that the induction hypothesis *IH* becomes applicable is a main challenge in automated mathematical induction [16,42,82].

We call the component of an induction theorem prover which checks the *validity* of an induction formula the *simplifier*. This component either is implemented as a (first-order) theorem prover (tailored for proving induction formulas) or as an interface to the user in case of an interactive (first-order) system. In this paper, we aim to supplement the simplifier with a learning component in the following way: Once the simplifier has computed a proof, this proof is *analyzed* and then *generalized* in a certain sense such that it can be *reused* subsequently. Before the simplifier is asked to prove another statement, now the system first looks for a previously computed proof of a *similar* statement and tries to reuse it. If the reuse fails, the simplifier has to compute an original proof for the new statement (as it must without a reuse facility). Otherwise (depending on the simplifier's implementation) either the *search* for a proof or *user interactions* are saved.

2. Reusing proofs—An example

The success of our approach is based on our techniques for *preparing* given proofs (by *proof analysis* and *generalization*) as well as by our techniques for proof reuse (by *retrieval* and *adaptation* methods).

We illustrate our proposal by an example. We assume that functions are defined by a finite set EQ of *defining equations*, and we are interested in verifying that some conjecture φ follows *inductively* from EQ, i.e., $\varphi \in \mathrm{Th}_{ind}(EQ)$ for the inductive theory $\mathrm{Th}_{ind}(EQ)$ of the equation set EQ. If the inductive validity of such a statement φ is verified, we may add φ to a set L of *lemmata*, and subsequent proofs are based on the set $AX := EQ \cup L$ of *axioms*. Now if $\psi \in \mathrm{Th}_{ind}(AX)$ is shown for a new conjecture ψ, then ψ is an inductive consequence of EQ since $\mathrm{Th}_{ind}(AX) = \mathrm{Th}_{ind}(EQ)$, and ψ may be also inserted into L, etc. (see, e.g., [82] for a more detailed account on induction theorem proving).

[2] Throughout this paper *induction* stands for *mathematical induction* and should not be confused with induction in the sense of machine learning. The reuse of previously computed induction schemas or generalizations are *not* subject of our proposal.

For proving a statement ψ by induction, i.e., to verify $\psi \in \mathrm{Th}_{ind}(EQ)$, a suitable *induction axiom* from $\mathrm{Th}_{ind}(EQ)$ is selected by well-known automated methods, cf., e.g., [82], from which a set of *induction formulas* I_ψ is computed for ψ such that $I_\psi \subseteq \mathrm{Th}_{ind}(AX)$ entails $\psi \in \mathrm{Th}_{ind}(EQ)$. For instance, let the functions plus, sum and app be defined by the following equations where 0 and s (respectively empty and add) are the constructors of the sort number (respectively list): [3]

(plus-1) $\mathsf{plus}(0, y) \equiv y$,

(plus-2) $\mathsf{plus}(\mathsf{s}(x), y) \equiv \mathsf{s}(\mathsf{plus}(x, y))$,

(sum-1) $\mathsf{sum}(\mathsf{empty}) \equiv 0$,

(sum-2) $\mathsf{sum}(\mathsf{add}(n, x)) \equiv \mathsf{plus}(n, \mathsf{sum}(x))$,

(app-1) $\mathsf{app}(\mathsf{empty}, y) \equiv y$,

(app-2) $\mathsf{app}(\mathsf{add}(n, x), y) \equiv \mathsf{add}(n, \mathsf{app}(x, y))$.

Now, e.g., the lemma

(lem-1) $\mathsf{plus}(\mathsf{plus}(x, y), z) \equiv \mathsf{plus}(x, \mathsf{plus}(y, z))$

can be easily proved by induction and therefore may be used as an axiom like any defining equation in subsequent deductions. We aim to prove conjectures as (lem-1) by reusing previously computed proofs of other lemmata. For instance consider the statement

$$\varphi[x, y] := \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(x, y)).$$

We prove the conjecture φ by induction upon the list-variable x and obtain two induction formulas, viz. the base formula φ_b and the step formula φ_s as

$$\varphi_b := \varphi[\mathsf{empty}, y],$$
$$\varphi_s := (\forall u\ \varphi[x, u]) \rightarrow \varphi[\mathsf{add}(n, x), y].$$

The following proof of the step formula φ_s is obtained by modifying the induction conclusion $\varphi[\mathsf{add}(n, x), y] =$

$$\mathsf{plus}(\mathsf{sum}(\mathsf{add}(n, x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n, x), y)) \quad \text{IC}$$

[3] When presenting examples, we usually omit universal quantifiers at the top level of formulas as well as the sort information for variables.

$$\Phi_s := (\forall u\ F(G(x), G(u)) \equiv G(H(x,u))) \rightarrow F(G(D(n,x)), G(y)) \equiv G(H(D(n,x), y))$$

$$C_s := \left\{ \begin{array}{ll} (1) & G(D(n,x)) \equiv F(n, G(x)) \\ (2) & H(D(n,x), y) \equiv D(n, H(x,y)) \\ (3) & F(F(x,y), z) \equiv F(x, F(y,z)) \end{array} \right\}$$

Fig. 1. The proof shell PS_s for the proof of φ_s.

in a backward chaining style, i.e., each statement is implied by the statement in the line below, where terms are underlined if they have been changed in the corresponding proof step:

$\mathsf{plus}(\mathsf{sum}(\mathsf{add}(n,x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n,x), y))$		IC
$\mathsf{plus}(\underline{\mathsf{plus}(n, \mathsf{sum}(x))}, \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n,x), y))$		(sum-2)
$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\underline{\mathsf{add}(n, \mathsf{app}(x,y))})$		(app-2)
$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \underline{\mathsf{plus}(n, \mathsf{sum}(\mathsf{app}(x,y)))}$		(sum-2)
$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \mathsf{plus}(n, \underline{\mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y))})$		IH
$\underline{\mathsf{plus}(n, \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y)))} \equiv \mathsf{plus}(n, \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y)))$		(lem-1)
$\underline{\mathrm{TRUE}}$		$X \equiv X$

Given such a proof, it is *analyzed* to distinguish its *relevant* features from its *irrelevant* parts. Relevant features are specific to the proof and are collected in a *proof catch* because "similar" requirements must be satisfied if this proof is to be reused later on. We consider features like the positions where equations are applied, induction conclusions and hypotheses, general laws as $X \equiv X$, etc. as irrelevant because they can always be satisfied. So the catch of a proof is a *subset* of the set of leaves of the corresponding proof tree. Analysis of the above proof yields (sum-2), (app-2), and (lem-1) as the catch. For example, all we have to know about plus for proving φ_s is its associativity, but not its semantics or *how* plus is computed.

Next the conjecture, the induction formula and the catch are *generalized*[4] for obtaining a *proof shell* which stores the essentials of the proof and serves as the base for reusing the proof subsequently. Generalization is performed by replacing function *symbols* by function *variables* denoted by capital letters F, G, H, etc., yielding the *schematic conjecture* $\Phi := F(G(x), G(y)) \equiv G(H(x,y))$ with the corresponding *schematic induction formula* Φ_s as well as the *schematic catch* C_s for our example, cf. Fig. 1.

Here we use the generalization replacement $\mathsf{plus} \mapsto F$, $\mathsf{sum} \mapsto G$, $\mathsf{app} \mapsto H$, $\mathsf{add} \mapsto D$, and therefore Eq. (1) of C_s corresponds to (sum-2), Eq. (2) to (app-2), and Eq. (3) to (lem-1).

[4] Not to be confused with *generalization* of a formula φ as a preprocessing for proving φ by induction.

$$\Phi_b := F(G(C), G(y)) \equiv G(H(C, y))$$

$$C_b := \left\{ \begin{array}{l} (4)\ G(C) \equiv E \\ (5)\ F(E, y) \equiv y \\ (6)\ H(C, y) \equiv y \end{array} \right\}$$

Fig. 2. The proof shell PS_b obtained from the proof of φ_b.

The *base* formula φ_b of our example is proved by

$\mathsf{plus}(\mathsf{sum}(\mathsf{empty}), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{empty}, y))$	φ_b
$\mathsf{plus}(\underline{0}, \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{empty}, y))$	(sum-1)
$\underline{\mathsf{sum}(y)} \equiv \mathsf{sum}(\mathsf{app}(\mathsf{empty}, y))$	(plus-1)
$\mathsf{sum}(y) \equiv \mathsf{sum}(\underline{y})$	(app-1)
$\underline{\mathrm{TRUE}}$	$X \equiv X$.

Proof analysis yields the catch $c_b := \{(\mathsf{sum}\text{-}1), (\mathsf{plus}\text{-}1), (\mathsf{app}\text{-}1)\}$, and using $\mathsf{plus} \mapsto F$, $\mathsf{sum} \mapsto G$, $\mathsf{app} \mapsto H$, $\mathsf{empty} \mapsto C$, $0 \mapsto E$, we obtain the proof shell PS_b of Fig. 2.

If a new statement ψ shall be proved, a set of induction formulas I_ψ is computed for ψ. Then for proving an induction formula $\psi_i \in I_\psi$ *by reuse*, it is tested whether some proof shell *PS* exists which *applies for* ψ_i, i.e., whether ψ_i is a (second-order) instance of the schematic induction formula of *PS*. If the test succeeds, the obtained (second-order) matcher is applied to the schematic catch of *PS*, and if all formulas of the instantiated schematic catch can be proved (which may necessitate further proof reuses), ψ_i is verified by reuse since the truth of an instantiated schematic catch implies the truth of its (correspondingly) instantiated schematic induction formula.

For example, assume that the new conjecture

$$\psi[x, y] := \mathsf{times}(\mathsf{prod}(x), \mathsf{prod}(y)) \equiv \mathsf{prod}(\mathsf{app}(x, y))$$

shall be proved, where times and prod are defined by the equations

(times-1)	$\mathsf{times}(0, y) \equiv 0$,
(times-2)	$\mathsf{times}(\mathsf{s}(x), y) \equiv \mathsf{plus}(y, \mathsf{times}(x, y))$,
(prod-1)	$\mathsf{prod}(\mathsf{empty}) \equiv \mathsf{s}(0)$,
(prod-2)	$\mathsf{prod}(\mathsf{add}(n, x)) \equiv \mathsf{times}(n, \mathsf{prod}(x))$.

The induction formulas computed for ψ are

$$\psi_b := \psi[\mathsf{empty}, y],$$

$$\psi_s := (\forall u\ \psi[x, u]) \rightarrow \psi[\mathsf{add}(n, x), y].$$

Obviously ψ_s is an instance of Φ_s with respect to the matcher $\pi_s := \{F/\mathsf{times}, G/\mathsf{prod}, H/\mathsf{app}, D/\mathsf{add}\}$, cf. Fig. 1. Hence we may reuse the proof of φ_s by instantiating the schematic catch C_s and subsequent verification of the resulting proof obligations:

$$\pi_s(C_s) = \left\{ \begin{array}{ll} (7) & \mathsf{prod}(\mathsf{add}(n,x)) \equiv \mathsf{times}(n, \mathsf{prod}(x)) \\ (8) & \mathsf{app}(\mathsf{add}(n,x), y) \equiv \mathsf{add}(n, \mathsf{app}(x,y)) \\ (9) & \mathsf{times}(\mathsf{times}(x,y), z) \equiv \mathsf{times}(x, \mathsf{times}(y,z)) \end{array} \right\}.$$

Formulas (7) and (8) are axioms, viz. (prod-2) and (app-2), and therefore are obviously true. So it only remains to prove the associativity of times (9) and, if successful, ψ_s is proved. Compared to a *direct* proof of ψ_s we have saved the user interactions respectively the search necessary to apply the right axioms in the right place (where the associativity of times must be verified in either case).[5] Note that the reuse *speculates* conjecture (9) *as a lemma* which is *required* for proving conjecture ψ. This lemma now is verified either directly or by calling the reuse procedure *recursively*.

To complete the proof of ψ, the base formula $\psi_b = \mathsf{times}(\mathsf{prod}(\mathsf{empty}), \mathsf{prod}(y)) \equiv \mathsf{prod}(\mathsf{app}(\mathsf{empty}, y))$ has to be verified, too. As this formula is an instance of Φ_b with respect to the matcher $\pi_b := \{F/\mathsf{times}, G/\mathsf{prod}, H/\mathsf{app}, C/\mathsf{empty}\}$, cf. Fig. 2, the schematic catch C_b is instantiated to

$$\pi_b(C_b) = \left\{ \begin{array}{ll} (10) & \mathsf{prod}(\mathsf{empty}) \equiv E \\ (11) & \mathsf{times}(E, y) \equiv y \\ (12) & \mathsf{app}(\mathsf{empty}, y) \equiv y \end{array} \right\}.$$

However, this schematic catch is only *partially* instantiated because the function variable E stemming from the function symbol 0 in the catch c_b is not replaced by the matcher π_b. This is because the function symbol 0 does not occur in ψ_b and consequently the function variable E does not occur in the schematic base formula Φ_b of the proof shell PS_b. We call such function variables of a proof shell *free* and we call all other function variables *bound* function variables.

A formula φ with a free function variable F is *true* iff *some* function exists such that the formula φ' obtained from φ by replacing F with this function is true. Thus, e.g., a true formula, viz. the axiom (prod-1), is obtained from (10) if E is replaced by $\mathsf{s}(0)$. Formally such replacements are represented by a second-order substitution like $\rho_b = \{E/\mathsf{s}(0)\}$, and ρ_b is called a *solution* (for the free function variables) if all formulas of the *totally* instantiated catch $\rho_b(\pi_b(C_b))$ are provable from AX: Here Eq. (12) is the axiom (app-1), equation $\rho_b(10)$ is the axiom (prod-1), and $\rho_b(11)$ simplifies to the speculated lemma $\mathsf{plus}(y, 0) \equiv y$ as the only remaining proof obligation.[6]

So generally, after finding a proof shell $PS = \langle \Phi, C \rangle$ which applies for a given conjecture ψ via some matcher π, i.e., $\pi(\Phi) = \psi$, a solution candidate ρ has to be computed from the

[5] We choose a very simple example to illustrate the essentials of our proposal. It should be obvious that a proof of φ and ψ can be computed by a state-of-the-art theorem prover without any search at all and therefore "reuse" offers no *real* savings in this case.

[6] Simplified conjectures are obtained by *symbolic evaluation*, cf. [82] and Section 7.3. For example, $\mathsf{s}(t_1) \equiv \mathsf{s}(t_2)$ is simplified to $t_1 \equiv t_2$ and $\mathsf{plus}(\mathsf{s}(t_1), t_2)$ is simplified to $\mathsf{s}(\mathsf{plus}(t_1, t_2))$.

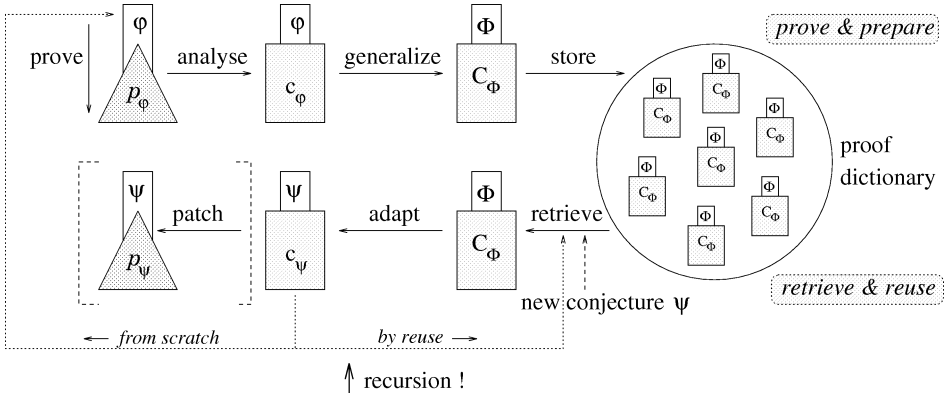

Fig. 3. The phases of the reuse process.

partially instantiated catch $\pi(C)$ in a subsequent *adaptation* step, and the resulting proof obligations from the *totally* instantiated catch $\rho(\pi(C))$ are subject to further verifications.

If one is also interested in a *proof* of ψ to be presented to a user or to be processed subsequently, it is not sufficient just to *instantiate* the schematic proof P of Φ (which is obtained by generalizing the proof p of φ) with the computed substitution $\tau := \rho \circ \pi$ because τ might destroy the structure of P. Therefore the instantiated proof $\tau(P)$ is *patched* (which always succeeds) by removing void respectively inserting additional inference steps for obtaining a proof p' of ψ. We do not discuss proof patching in this paper and refer to [53,57] for details.

Fig. 3 illustrates the overall organization of our approach for reusing proofs, and we discuss the steps of the procedure in the subsequent sections.

3. Proof analysis and generalization

In this section a formal base for proof analysis and generalization is developed. We use a first-order language to represent axioms and the formulas derived from them:

Definition 3.1 (*Symbols, terms, formulas*). Let

$$\mathrm{SIG} = \bigcup_{n\in\mathbb{N}} \mathrm{SIG}_n$$

be a signature for function symbols, i.e., each set SIG_n holds function symbols of arity n, and let VAR be a set of variable symbols. Then $\mathcal{T}(\mathrm{SIG}, \mathrm{VAR})$ denotes the set of all well-formed terms over $\mathrm{SIG} \cup \mathrm{VAR}$. An equation (over $\mathrm{SIG} \cup \mathrm{VAR}$) is an expression of the form $t_1 \equiv t_2$, where $t_1, t_2 \in \mathcal{T}(\mathrm{SIG}, \mathrm{VAR})$, $\mathcal{EQ}(\mathrm{SIG}, \mathrm{VAR})$ is the set of all well-formed equations, and $\mathcal{F}(\mathrm{SIG}, \mathrm{VAR})$ is the set of all well-formed formulas, build with equations and the predicate symbols TRUE and FALSE as atomic formulas by using the usual connectives and quantifiers.

The first-order language over the signature $\Sigma = \{0, s, empty, add, plus, times, sum, prod, app, \ldots\}$ in which we write axioms and conjectures is called the *object language*:

Definition 3.2 (*Object language*). Let Σ be a fixed signature for function symbols and let $\mathcal{V}$ be a fixed set of variable symbols. Then Σ is called the *object* signature, $\mathcal{T}(\Sigma, \mathcal{V})$ is the set of all *object terms* and $\mathcal{F}(\Sigma, \mathcal{V})$ is the *object language*.

3.1. Simple proof analysis

Since we want to investigate the principles of proof analysis and reuse, we confine ourselves here with unconditional equations as the only conjectures to be proved. Hence the set AX of *axioms* (i.e., defining equations and lemmata) from which inferences are drawn is a set of (universally closed) equations of the object language. However, when proving a conjecture we have to consider induction formulas of the form $IH \to IC$, cf. Section 2: Such formulas $H \to C$ are called *sequents*, where $H := H_1 \wedge \cdots \wedge H_n$ is the set of *hypotheses*[7] each of which has the form $\forall u^*\ t_1 \equiv t_2$, the *conclusion* C has the form $s_1 \equiv s_2$, and the free variables x^* of $H \to C$ are implicitly assumed to be universally quantified.[8]

To formalize the *simple* proof analysis we define a calculus in which we prove (base and step) formulas. Each rule of this calculus is built from a "conventional" inference rule by stipulating in addition which formula has to be remembered as a "relevant feature" if the particular inference rule is used in a proof. Hence the rules are applied to expressions of the form $\langle \varphi, A \rangle$, where φ is a sequent and $A \subseteq AX$, called the *accumulator*, holds the catch collected so far.

Definition 3.3 (*Simple analysis calculus*). The *simple analysis calculus* consists of the following inference rules operating on pairs $\langle \varphi, A \rangle$ of a sequent φ and an accumulator $A \subseteq AX$, with respect to a set of equational axioms AX. Here θ is a substitution and p is a position in the equation C:[9]

- *Reflexivity*

$$\frac{\langle \forall x^*\ H \to t \equiv t, A \rangle}{\langle \mathrm{TRUE}, A \rangle}.$$

- *AX-replacement*

$$\frac{\langle \forall x^*\ H \to C, A \rangle}{\langle \forall x^*\ H \to C[p \leftarrow \theta(r)], A \cup \{\forall u^*\ l \equiv r\} \rangle}$$

if $\forall u^*\ l \equiv r \in AX$, $l \notin u^*$, $C|_p = \theta(l)$, $\mathcal{V}(r) \subseteq \mathcal{V}(l)$, and $\mathrm{dom}(\theta) = \mathcal{V}(l)$.

[7] We do not distinguish between the *formula* H and the *set* H which contains the members of the conjunction H.

[8] Instead of operating on sequents $H \to C$, a calculus can also be defined operating on the equation C with respect to a set of additional "local" equational hypotheses H, so that only equations are modified by the rules of the calculus. However we stick to the use of sequents because this eases the presentation.

[9] We assume familiarity with the standard notions of equality reasoning, like positions, subterm replacement, etc. [24]. Also symmetry of $\equiv$ is implicitly assumed.

- *HYP-replacement*

$$\frac{\langle \forall x^*\ H \to C, A\rangle}{\langle \forall x^*\ H \to C[p \leftarrow \theta(r)], A\rangle}$$

if $\forall u^*\ l \equiv r \in H$, $l \notin u^*$, $C|_p = \theta(l)$, $(\mathcal{V}(r) \setminus \mathcal{V}(l)) \cap u^* = \emptyset$, and $\mathrm{dom}(\theta) = \mathcal{V}(l) \cap u^*$.

A sequence $\langle\langle \varphi_1, A_1\rangle, \ldots, \langle \varphi_n, A_n\rangle\rangle$ of pairs of sequents φ_i and accumulators $A_i \subseteq AX$ is a *derivation* in the analysis calculus from a set AX of equational axioms iff $\langle \varphi_{i+1}, A_{i+1}\rangle$ results from applying one of the rules to $\langle \varphi_i, A_i\rangle$ for each $i \in \{1, \ldots, n-1\}$. Derivability is denoted by $\langle \varphi_1, A_1\rangle \vdash^{\mathrm{a}}_{AX} \langle \varphi_n, A_n\rangle$, where AX may be omitted if appropriate.

The requirement $\mathrm{dom}(\theta) \subseteq u^*$ guarantees that no *free* variables from $\forall u^*\ l \equiv r$ are instantiated if the applied equation is a hypothesis, while this requirement is void if the applied equation is an axiom (as this is always universally closed). Analogously the requirements $(\mathcal{V}(r) \setminus \mathcal{V}(l)) \cap u^* = \emptyset$ respectively $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ ensure that no new variables are introduced in the sequent. The additional requirement $\mathrm{dom}(\theta) \subseteq \mathcal{V}(l)$ is demanded to keep the substitution θ minimal, i.e., no variables are replaced unnecessarily.

The replacement rules differ only in updating the accumulator component: If the applied equation $\forall u^*\ l \equiv r$ is an *axiom* it has to be recorded in A for obtaining the catch, but if the applied equation is a *hypothesis* it is irrelevant for the learning step and A remains unchanged.

Example 3.4 (*Simple proof analysis*). Consider the expression $\langle H \to g(f(c, h(y))) \equiv \ldots, \{\ldots\}\rangle$ and the equation $E := \forall x\ \ f(c, x) \equiv h(x)$. If $E \in H$, then the expression $\langle H \to g(h(h(y))) \equiv \ldots, \{\ldots\}\rangle$ is obtained by *HYP*-replacement. But if $E \in AX$, then AX-replacement yields $\langle H \to g(h(h(y))) \equiv \ldots, \{E, \ldots\}\rangle$.

The calculus for simple proof analysis yields a finite subset of axioms for the derivation of a sequent and is sound:

Theorem 3.5 (Soundness of $\vdash^{\mathrm{a}}_{AX}$). *Let $AX, A \subset \mathcal{F}(\Sigma, \mathcal{V})$ be sets of universally closed equations and let φ be a sequent in $\mathcal{F}(\Sigma, \mathcal{V})$ such that $\langle \varphi, \emptyset\rangle \vdash^{\mathrm{a}}_{AX} \langle \mathrm{TRUE}, A\rangle$. Then*

(i) *$A \subset AX$, $|A| < \infty$, and $\langle \varphi, \emptyset\rangle \vdash^{\mathrm{a}}_{A} \langle \mathrm{TRUE}, A\rangle$,*
(ii) *$A \models \varphi$, and*
(iii) *$AX \models \varphi$.*

Proof. (i) $A \subseteq AX$ is obvious from the definition of A by the rules of the calculus and $|A| < \infty$ since each derivation is finite. Finally each derivation from AX also is a derivation from A, since no axioms from $AX \setminus A$ are used in the derivation.

(ii) φ can be inferred from A, cf. (i), hence $A \models \varphi$ as the reflexivity rule as well as both replacement rules are sound.

(iii) Follows from (i) and (ii) by the monotonicity of semantical entailment. □

Given a formula ψ and a set of induction formulas $\{\psi_0, \ldots, \psi_n\}$ for ψ, we try to infer $\langle \psi_i, \emptyset\rangle \vdash^{\mathrm{a}}_{AX} \langle \mathrm{TRUE}, A_i\rangle$ for each i in our calculus. If successful,

$$AX \supseteq A_0 \cup \cdots \cup A_n \models \{\psi_0, \ldots, \psi_n\} \models_{ind} \psi,$$

i.e., ψ is proved, and for each i an accumulator A_i is obtained which holds the catch of the proof of ψ_i.

3.2. Generalization

Given a formula φ and a catch c of a proof of φ from AX, we reuse this proof for verification of another formula ψ by a consistent replacement of the function symbols in $c \cup \{\varphi\}$ with other function symbols or terms yielding a set of equations c' and a formula $\varphi' = \psi$ such that $AX \models c'$ holds. For example, the proof of the base formula

$$\mathsf{plus}(\mathsf{sum}(\mathsf{empty}), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{empty}, y))$$

is reused for proving the base formula

$$\mathsf{times}(\mathsf{prod}(\mathsf{empty}), \mathsf{prod}(y)) \equiv \mathsf{prod}(\mathsf{app}(\mathsf{empty}, y))$$

using the replacements $\mathsf{plus} \mapsto \mathsf{times}$, $\mathsf{sum} \mapsto \mathsf{prod}$ and $\mathsf{0} \mapsto \mathsf{s}(\mathsf{0})$, cf. Section 2. However, in order to ease the presentation and to avoid formal clutter we prefer an indirect way of formulating those function symbol replacements: In a *generalization step*, the function and variable symbols of $c \cup \{\varphi\}$ are consistently replaced by symbols from a signature Ω and a set $\mathcal{U}$ of variable symbols yielding the *schematic catch* C and the *schematic conjecture* Φ. These formulas are members of another first-order language, viz. the *schematic language* $\mathcal{F}(\Omega, \mathcal{U})$. If a proof is to be reused for proving some new conjecture ψ, the symbols from $\Omega \cup \mathcal{U}$ are replaced by symbols from $\Sigma \cup \mathcal{V}$ to match the schematic conjecture Φ with ψ. For example, for the example above, we use $\mathsf{plus} \mapsto F$, $\mathsf{sum} \mapsto G$, $\mathsf{empty} \mapsto C$, $\mathsf{app} \mapsto H$, $\mathsf{0} \mapsto E$, $y \mapsto u$ in the generalization step and obtain the schematic conjecture $\Phi = F(G(C), G(u)) \equiv G(H(C, u))$, and $F \mapsto \mathsf{times}$, $G \mapsto \mathsf{prod}$, $C \mapsto \mathsf{empty}$, $H \mapsto \mathsf{app}$, $E \mapsto \mathsf{s}(\mathsf{0})$, $u \mapsto y$ then is used in the reuse step, cf. Section 2.

Definition 3.6 (*Schematic language*). Let Ω be a signature for function symbols and let $\mathcal{U}$ be a set of variable symbols such that $\Sigma \cap \Omega = \mathcal{V} \cap \mathcal{U} = \emptyset$. Then $\mathcal{T}(\Omega, \mathcal{U})$ is the set of *schematic terms*, $\mathcal{EQ}(\Omega, \mathcal{U})$ is the set of *schematic equations*, and $\mathcal{F}(\Omega, \mathcal{U})$ is the *schematic language*, i.e., the set of all *schematic formulas*.

Function symbols from the schematic signature Ω are denoted by capital letters $F, G, \ldots$ and referred to as *function variables* subsequently (which indicates the intended replacements by function *symbols* from the object signature Σ). The variables in $\mathcal{U}$ are also called *schematic variables*. Variables and function symbols are formally replaced by schematic variables and function variables using *generalization functions*:

Definition 3.7 (*Generalization functions*). A *generalization function* γ is an injective and partial function $\gamma : \Sigma \cup \mathcal{V} \to \Omega \cup \mathcal{U}$ with finite domain $\mathrm{dom}(\gamma)$ which maps function symbols to function variables of the same arity and maps variables to schematic variables, i.e., $f \in \mathrm{dom}(\gamma) \cap \Sigma_n$ implies $\gamma(f) \in \Omega_n$ and $x \in \mathrm{dom}(\gamma) \cap \mathcal{V}$ implies $\gamma(x) \in \mathcal{U}$. A generalization function γ is homomorphically extended to object formulas, i.e., $\gamma(\varphi) \in \mathcal{F}(\Omega, \mathcal{U})$ for each $\varphi \in \mathcal{F}(\Sigma, \mathcal{V})$ such that $\Sigma(\varphi) \cup \mathcal{V}(\varphi) \subseteq \mathrm{dom}(\gamma)$.

A generalization function γ maps an object formula φ to a schematic formula $\gamma(\varphi)$. The inverse mapping, i.e., the mapping of a schematic formula Φ to an object formula $\pi(\Phi)$, also called a *total instantiation* of Φ, is achieved by *schematic substitutions* and *variable renamings*, cf. Section 4. We use the schematic language $\mathcal{F}(\Omega, \mathcal{U})$ to represent "proof ideas" in form of *proof shells*:

Definition 3.8 (*Proof shells*). A pair $PS = \langle \Phi, C \rangle$ consisting of a schematic formula $\Phi \in \mathcal{F}(\Omega, \mathcal{U})$ and a finite set $C \subseteq \mathcal{F}(\Omega, \mathcal{U})$ of schematic formulas is a *proof shell* iff $C \models \Phi$.

Proof shells constitute the elementary building blocks for proof reuse. They are computed by generalizing an object formula φ and its proof catch c to a proof shell $\langle \Phi, C \rangle$:

Theorem 3.9 (Proof shells from simple analysis). *Let* $A \cup \{\varphi\} \subset \mathcal{F}(\Sigma, \mathcal{V})$ *such that* $\langle \varphi, \emptyset \rangle \vdash^{\mathrm{a}}_{A} \langle \mathrm{TRUE}, A \rangle$ *and let* γ *be a generalization function such that* $\gamma(A \cup \{\varphi\}) \subset \mathcal{F}(\Omega, \mathcal{U})$. *Then* $\langle \gamma(\varphi), \gamma(A) \rangle$ *is a proof shell.*

Proof. $A \models \varphi$ by Theorem 3.5(ii) and consequently $\gamma(A) \models \gamma(\varphi)$ since γ is a one-to-one symbol renaming. □

By Theorem 3.9 each proof of an object formula in the simple analysis calculus can be generalized to a proof shell with the *schematic conjecture* $\gamma(\varphi)$ and the *schematic catch* $\gamma(A)$. See Section 2 for examples of proof shells obtained according to Theorem 3.9.

4. Instantiating proof shells

Proof analysis and generalization define the *proof-&-prepare* phase of our reuse procedure, cf. Fig. 3, which transforms a proof into a reusable data structure, viz. a proof shell. In the following we are concerned with the *retrieve-&-reuse* phase, cf. Fig. 3, which aims at the selection and instantiation of a proof shell for a given verification task, such that verifiable proof obligations are obtained. Before we discuss the retrieval and adaptation steps in Section 7, the concept of *admissible* and *total* instantiations of proof shells is introduced.

A conjecture ψ is verified by reuse, if a proof shell $\langle \Phi, C \rangle$ and a certain substitution π exists, such that $\pi(C \cup \{\Phi\}) \subseteq \mathcal{F}(\Sigma, \mathcal{V})$, $\pi(\Phi) = \psi$ and $AX \models \pi(C)$, see Section 2 for examples. Such a substitution π is computed *iteratively* as $\pi_n \circ \cdots \circ \pi_1 \circ \pi_0$, i.e., one computes a substitution π_0 by considering Φ, π_1 by considering $\pi_0(C)$, π_2 by considering $\pi_1(\pi_0(C))$ etc., until eventually $\pi_n(\ldots \pi_1(\pi_0(C \cup \{\Phi\}))\ldots) \subseteq \mathcal{F}(\Sigma, \mathcal{V})$ and $\pi_n(\ldots \pi_1(\pi_0(\Phi))\ldots) = \psi$. Finally $AX \models \pi_n(\ldots \pi_1(\pi_0(C))\ldots)$ is verified, and if successful $AX \models \psi$ is proved. We call the formulas in the intermediate steps, i.e., the formulas in $\pi_0(C)$, $\pi_1(\pi_0(C))$, etc. *mixed formulas* since they neither belong to the *schematic* nor to the *object* language:

Definition 4.1 (*Mixed language*). $\mathcal{T}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$ is the set of *mixed terms* and $\mathcal{F}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$ is the *mixed language*, i.e., the set of all *mixed formulas*.

Since $\mathcal{F}(\Sigma,\mathcal{V}) \cup \mathcal{F}(\Omega,\mathcal{U}) \subseteq \mathcal{F}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$, each *object* formula and each *schematic* formula also is a *mixed formula*. In a mixed (or schematic) expression, schematic variables are successively replaced with object variables and function variables are successively replaced with (a composition of) function symbols by application of certain substitutions such that eventually an expression of the object language is obtained (after creating mixed expressions in the intermediate steps).

Definition 4.2 (*Schematic renamings, schematic substitutions*). A *schematic variable renaming* is an injective mapping $\nu : \mathcal{U} \to \mathcal{V}$. The application of ν to a mixed term or formula is defined as the homomorphical extension of ν.

A *schematic substitution* $\pi : \Omega \to \mathcal{T}(\Sigma \cup \Omega, \mathcal{W})$ is a partial function with finite domain $\mathrm{dom}(\pi)$ such that $\pi(F) \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{W}_F)$ for each $F \in \mathrm{dom}(\pi)$. Here $\mathcal{W} := \bigcup_{F\in\Omega} \mathcal{W}_F$ is the set of *parameter variables*, where $\mathcal{W}_F := \{F_1, F_2, \ldots, F_n\}$ for $F \in \Omega_n$ and $\mathcal{W}_F \cap \mathcal{W}_G = \emptyset$ for $F \neq G$. $\mathcal{T}(\Sigma \cup \Omega, \mathcal{W})$ is called the set of *functional terms*. The *application* of π to a mixed term $M \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$ is inductively defined by

$$\begin{aligned}
&\pi(z) := z && \text{if } z \in \mathcal{V} \cup \mathcal{U},\\
&\pi(F(M_1,\ldots,M_n)) := F(\pi(M_1),\ldots,\pi(M_n)) && \text{if } F \notin \mathrm{dom}(\pi),\\
&\pi(F(M_1,\ldots,M_n)) := \sigma(\pi(F)) && \text{if } F \in \mathrm{dom}(\pi),
\end{aligned}$$

where $\sigma := \{F_1/\pi(M_1), \ldots, F_n/\pi(M_n)\}$ replaces the parameter variables F_i in $\pi(F) \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{W}_F)$. The application of π to a mixed formula is defined as the homomorphical extension of π.

A schematic substitution π is usually written as a finite set of replacement pairs such that $F/M \in \pi$ iff $F \in \mathrm{dom}(\pi)$ and $M = \pi(F)$. A functional term $M \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{W}_F)$ corresponds to the λ-term $\lambda F_1, \ldots, F_n.M$ from the λ-calculus, and a schematic substitution is also called a *pure and closed second-order substitution* because variables from $\mathcal{V} \cup \mathcal{U}$ are neither replaced nor introduced.[10] We usually write w_i (instead of F_i) for the ith formal parameter in a functional term when we define schematic substitutions in examples and F is obvious from the context. F/f with $F \in \Omega_n$ and $f \in \Sigma_n$ is an abbreviation for the replacement pair $F/f(w_1,\ldots,w_n)$ which retains the structure of terms during instantiations, i.e., $\pi(F(t_1,\ldots,t_n)) = f(\pi(t_1),\ldots,\pi(t_n))$. More complex instantiations like, e.g., $\pi = \{G/\mathsf{plus}(w_2, \mathsf{len}(w_1)),\ H/F(w_1,w_1)\}$ for $G, H \in \Omega_2$ can severely change the structure of terms since whole subterms can be introduced, multiplied, or deleted, as, e.g., $\pi(G(H(x, \mathsf{sum}(\mathsf{app}(k,l))), \mathsf{s}(y))) = \mathsf{plus}(\mathsf{s}(y), \mathsf{len}(F(x,x)))$.

Now the reuse of proofs by instantiation of proof shells can be formulated:

Theorem 4.3 (Reuse theorem). *Let $\langle \Phi, C\rangle$ be a proof shell, ν be a schematic variable renaming, and π be a schematic substitution such that $\nu(\pi(C \cup \{\Phi\})) \subseteq \mathcal{F}(\Sigma,\mathcal{V})$. Then $\nu(\pi(C)) \models \nu(\pi(\Phi))$.*

Proof. $C \models \Phi$ by Definition 3.8, and consequently $\nu(\pi(C)) \models \nu(\pi(\Phi))$ since entailment is invariant with respect to instantiations, cf. [50]. □

[10] The notation using parameter variables is borrowed from [37].

5. Sortal reasoning

The domain of discourse in theorem proving often is *many-sorted*, as, e.g., in induction theorem proving statements about *natural numbers, linear lists, trees*, etc. are considered. Sortal reasoning is incorporated into logic by assigning a *rangesort* from a set $\mathcal{S}$ of sort symbols, say $\mathcal{S} = \{\mathsf{bool}, \mathsf{nat}, \mathsf{list}, \ldots\}$, to variable and function symbols, and also assigning a *domainsort* from $\mathcal{S}$ to each argument position of a function symbol. Sorts are also assigned to terms: The *sort* of a term t is the rangesort of t if t is a variable, and it is the rangesort of f, if $t = f(\ldots)$. A term t is *well-sorted* iff for each subterm $f(t_1, \ldots, t_n)$ of t the sort of each t_i coincides with the domainsort of f at position i. A substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is *well-sorted* iff the sort of each t_i coincides with the sort of x_i and each t_i is well-sorted. Now in many-sorted logic one demands that

(1) only well-sorted terms are used,
(2) t_1 and t_2 have the same sort for each equation $t_1 \equiv t_2$ in a formula, and
(3) only well-sorted substitutions are used in a deduction.

Different from non-flat sort hierarchies, where an order relation is imposed on $\mathcal{S}$ thus representing inclusion of sets, cf. [79], a flat many-sorted framework influences deduction only when matchers (or unifiers) are computed for a *variable* replacement: A term t in a formula may be replaced by a term $\theta(r)$ using the equation $l \equiv r$ and a matcher θ of l and t, only if θ is well-sorted. This is guaranteed if l is not a variable, because all terms are well-sorted. However, the sorts of l and t must be explicitly compared, if l is a variable. Otherwise, θ may be ill-sorted and consequently ill-sorted terms may be formed in a deduction, as, e.g., a term $\mathsf{plus}(a, b)$ may be replaced by $\mathsf{rev}(\mathsf{rev}(\mathsf{plus}(a, b)))$ using the equation $x \equiv \mathsf{rev}(\mathsf{rev}(x))$ and an ill-sorted matcher which replaces a list-variable x by a nat-term $\mathsf{plus}(\ldots)$.

Now in order to guarantee a sound reuse in a *many-sorted logic*, information about the sorts of "locally" quantified variables l in equations $l \equiv r$ used from left to right in a proof must be memorized in the proof catch and properly considered on subsequent reuse attempts, such that provability of a statement for which the reuse succeeds is guaranteed in fact. Since this involves an awkward formal machinery for the rare case of pure variable replacements in a proof, cf. [52], we do not allow such variable replacements in the analysis calculus. Consequently, a proof catch is *independent* of sortal information.

However, when instantiating a proof shell, it has to be guaranteed that the intermediate mixed formulas have at least one *well-sorted ground instance*. Consider, for instance, a partially instantiated catch containing the mixed formulas

(1) $F(G(k)) \equiv \mathsf{s}(\mathsf{sum}(k))$, and
(2) $F(n) \equiv H(n)$,

where k is a list-variable and n is a variable of sort nat. The substitution $\pi_1 := \{F/\mathsf{s},\ G/\mathsf{sum}\}$ solves (1) and yields $\mathsf{s}(n) \equiv H(n)$ as a further instantiation of the remaining catch formula (2). The substitution $\pi_2 := \{F/\mathsf{s}(\mathsf{sum}(w_1)),\ G/w_1\}$ also solves (1) and here $\mathsf{s}(\mathsf{sum}(n)) \equiv H(n)$ results from (2). But $\mathsf{s}(\mathsf{sum}(n))$ is ill-sorted (as n is a nat-variable but $\mathsf{sum} : \mathsf{list} \to \mathsf{nat}$), hence π_2 had not been considered as a solution substitution for (1). Since the selection of a wrong substitution may be recognized only after several instantiation steps, machine resources are wasted on reuse attempts and backtracking mechanisms are required.

When forming a totally instantiated catch, backtracking can be avoided if sorts are assigned also to the *schematic variable symbols* and the *function variables*. Then we call a set Π of schematic variable renamings and substitutions *admissible* for a set C of mixed formulas, if a sort assignment exists such that each variable renaming and substitution in Π and each formula in C is *well-sorted*.

For instance, $F : \mathsf{nat} \to \mathsf{nat}$, $G : \mathsf{list} \to \mathsf{nat}$ and $H : \mathsf{nat} \to \mathsf{nat}$ is a sort assignment such that $\{(1), (2)\}$ as well as π_1 are well-sorted, and therefore $\{\pi_1\}$ is admissible for $\{(1), (2)\}$. But $\{\pi_2\}$ is not admissible for $\{(1), (2)\}$, since the well-sortedness of $\{(1), (2)\}$ demands $F : \mathsf{nat} \to \mathsf{nat}$, whereas the well-sortedness of π_2 demands $F : \mathsf{list} \to \mathsf{nat}$.

Admissibility of a set Π of substitutions for a set C of mixed formulas is easily tested by computing an equational system $E(\Pi, C)$ over sort symbols from Π and C and subsequent verification, that no different sorts, as, e.g., nat and list above, are identified by $E(\Pi, C)$, see [84] for formal details. For the reuse of proofs in a sorted logic, the Reuse Theorem 4.3 is reformulated as:

Corollary 5.1 (Reuse theorem for sorted logic). *Let $\langle \Phi, C \rangle$ be a proof shell, ν be a schematic variable renaming, and π be a schematic substitution such that*

(1) *$\{\pi, \nu\}$ is admissible for $C \cup \{\Phi\}$, and*

(2) *$\nu(\pi(C \cup \{\Phi\})) \subseteq \mathcal{F}(\Sigma, \mathcal{V})$.*

Then $\nu(\pi(C)) \models_S \nu(\pi(\Phi))$, where $\models_S$ denotes semantical entailment of sorted logic, cf. [29].

Proof. Follows from Theorem 4.3, cf. [50]. □

6. Refined proof analysis

Corollary 5.1 provides the logical base for our proposal of reusing proofs: For a given conjecture ψ, some proof shell $\langle \Phi, C \rangle$, some renaming ν and some schematic substitution π must be found which meet requirements (1) and (2) of the Reuse Theorem and also satisfy $\psi = \nu(\pi(\Phi))$. Then $AX \models \nu(\pi(\psi'))$ is verified (either directly or "by reuse" again) for each $\psi' \in C$, and if successful $AX \models \psi$ is proved.

Since our proof shells are obtained from (analyzed) proofs, cf. Theorem 3.9, the success of proof reuse directly depends on the generality of what has been learned from a given proof:

Example 6.1 (*Limitations of simple analysis*). Let $AX = \{\mathsf{f(a)} \equiv \mathsf{a},\ \mathsf{g(a)} \equiv \mathsf{b},\ \mathsf{g(b)} \equiv \mathsf{a}\}$ and consider the conjecture $\varphi = \mathsf{f(f(a))} \equiv \mathsf{a}$. Then

$$
\begin{aligned}
& \langle \mathsf{f(f(a))} \equiv \mathsf{a}, \emptyset \rangle \\
\vdash^{\mathsf{a}}_{AX}\ & \langle \mathsf{f(a)} \equiv \mathsf{a}, \{\mathsf{f(a)} \equiv \mathsf{a}\} \rangle \\
\vdash^{\mathsf{a}}_{AX}\ & \langle \mathsf{a} \equiv \mathsf{a}, \{\mathsf{f(a)} \equiv \mathsf{a}\} \rangle \\
\vdash^{\mathsf{a}}_{AX}\ & \langle \mathrm{TRUE}, \{\mathsf{f(a)} \equiv \mathsf{a}\} \rangle
\end{aligned}
$$

and the proof shell $PS = \langle F(F(A)) \equiv A, \{F(A) \equiv A\} \rangle$ is obtained from this proof. But PS does not apply for the conjecture $\psi_1 = \mathsf{f(g(b))} \equiv \mathsf{a}$ since the function variable F cannot

be replaced both by f and g (and the function variable A cannot be replaced both by b and a), and the reuse fails for ψ_1. *PS* applies for the conjecture $\psi_2 = \mathsf{g}(\mathsf{g}(\mathsf{a})) \equiv \mathsf{a}$, hence the instantiated catch is computed as $\{\mathsf{g}(\mathsf{a}) \equiv \mathsf{a}\}$. But $AX \not\models \mathsf{g}(\mathsf{a}) \equiv \mathsf{a}$, hence the reuse fails also for ψ_2.

One possible reason for the failed reuses is that each conjecture requires an original proof which differs in its structure from the proof of φ. But it may also be the case that what has been learned is simply not enough, so the reuse fails only for this reason. The latter is true for both conjectures in the example and we improve our technique so that the reuse eventually is successful.

The key idea for the improvement is to distinguish *different occurrences* of function symbols in the conjecture and the catch of the conjecture's proof, which may be *replaced* by (a composition of) *different function symbols* without spoiling the soundness of the proof. This yields a *more general* proof shell, since *different occurrences* of a function symbol at the object level are generalized to *different function variables* at the schematic level when the proof shell is formed.

Example 6.2 (*Example* 6.1 *continued*). We may recover from the reuse failures since both φ and AX have several occurrences of the function symbols f and a which can be separated into *different* function symbols without spoiling the soundness of the derivation: We may use $AX^* = \{\mathsf{f}'(\mathsf{a}) \equiv \mathsf{a}'',\ \mathsf{f}(\mathsf{a}'') \equiv \mathsf{a}'\}$ to prove $\varphi^* = \mathsf{f}(\mathsf{f}'(\mathsf{a})) \equiv \mathsf{a}'$ by

$$
\begin{aligned}
& \langle \mathsf{f}(\mathsf{f}'(\mathsf{a})) \equiv \mathsf{a}', \emptyset \rangle \\
\vdash^{\mathrm{a}}_{AX}\ & \langle \mathsf{f}(\mathsf{a}'') \equiv \mathsf{a}', \{\mathsf{f}'(\mathsf{a}) \equiv \mathsf{a}''\} \rangle \\
\vdash^{\mathrm{a}}_{AX}\ & \langle \mathsf{a}' \equiv \mathsf{a}', \{\mathsf{f}'(\mathsf{a}) \equiv \mathsf{a}'',\ \mathsf{f}(\mathsf{a}'') \equiv \mathsf{a}'\} \rangle \\
\vdash^{\mathrm{a}}_{AX}\ & \langle \mathrm{TRUE}, \{\mathsf{f}'(\mathsf{a}) \equiv \mathsf{a}'',\ \mathsf{f}(\mathsf{a}'') \equiv \mathsf{a}'\} \rangle
\end{aligned}
$$

and obtain the proof shell $PS^* = \langle F(F'(A)) \equiv A', \{F'(A) \equiv A'',\ F(A'') \equiv A'\} \rangle$ from this derivation.

PS^* applies for the conjecture ψ_1 via the matcher $\pi_1 = \{F/\mathsf{f},\ F'/\mathsf{g},\ A/\mathsf{b},\ A'/\mathsf{a}\}$, hence the partially instantiated catch C_1 is computed as $\{\mathsf{g}(\mathsf{b}) \equiv A'',\ \mathsf{f}(A'') \equiv \mathsf{a}\}$. Since $\rho_1 = \{A''/\mathsf{a}\}$ is a solution for the free function variable in C_1, i.e., $\rho_1(\pi_1(C_1)) = \{\mathsf{g}(\mathsf{b}) \equiv \mathsf{a},\ \mathsf{f}(\mathsf{a}) \equiv \mathsf{a}\} \subseteq AX$, ψ_1 is proved by reuse based on the proof shell PS^*.

PS^* applies also for ψ_2 via the matcher $\pi_2 = \{F/\mathsf{g},\ F'/\mathsf{g},\ A/\mathsf{a},\ A'/\mathsf{a}\}$, hence the partially instantiated catch C_2 is computed as $\{\mathsf{g}(\mathsf{a}) \equiv A'',\ \mathsf{g}(A'') \equiv \mathsf{a}\}$. Since $\rho_2 = \{A''/\mathsf{b}\}$ is a solution for the free function variable in C_2, i.e., $\rho_2(\pi_2(C_2)) = \{\mathsf{g}(\mathsf{a}) \equiv \mathsf{b},\ \mathsf{g}(\mathsf{b}) \equiv \mathsf{a}\} \subseteq AX$, also ψ_2 is proved by reuse based on the proof shell PS^*.

Since the original derivation of φ from AX does not expose the possible separation of the occurrences of f and a, all occurrences of f are generalized to one function variable F and all occurrences of a are generalized to one function variable A in the proof shell *PS*, and therefore the reuse fails for ψ_1 and ψ_2. So a remedy to the problem is to recognize *different occurrences* of function symbols in a derivation which may be generalized to *different function variables* when the proof shell is constructed.

To this effect, we distinguish in a first step the different occurrences of each function symbol in the conjecture by supplying the function symbols with different superscripts from $\mathbb{N}$ yielding, e.g., $\varphi' = \mathsf{f}^1(\mathsf{f}^2(\mathsf{a}^1)) \equiv \mathsf{a}^2$. We call function symbols with superscripts *indexed* function symbols and we call f the *root* and i the *index* of an indexed function symbol f^i. We also supply the function symbols of the axioms with unique indices, where we assume infinitely many *differently indexed copies* of each axiom in the indexed axiom set, and, e.g., $AX' = \{\mathsf{f}^3(\mathsf{a}^3) \equiv \mathsf{a}^4,\ \mathsf{f}^4(\mathsf{a}^5) \equiv \mathsf{a}^6,\ \mathsf{f}^5(\mathsf{a}^7) \equiv \mathsf{a}^8,\ \mathsf{f}^6(\mathsf{a}^9) \equiv \mathsf{a}^{10}, \ldots\}$ is obtained such that no indexed function symbol has more than one occurrence in the indexed axiom set and the indexed conjecture.

In the next step, the function symbols in the *derivation* are supplied with corresponding indices, where we demand that no indexed axiom is applied more than once in a derivation. This does not restrict derivability, since we have infinitely many indexed copies at our disposal, and we obtain

$$\begin{aligned}
&\langle \mathsf{f}^1(\mathsf{f}^2(\mathsf{a}^1)) \equiv \mathsf{a}^2, \emptyset\rangle\\
\vdash\ &\langle \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2, \{\mathsf{f}^3(\mathsf{a}^3) \equiv \mathsf{a}^4\}\rangle\\
\vdash\ &\langle \mathsf{a}^6 \equiv \mathsf{a}^2, \{\mathsf{f}^3(\mathsf{a}^3) \equiv \mathsf{a}^4,\ \mathsf{f}^4(\mathsf{a}^5) \equiv \mathsf{a}^6\}\rangle\\
\vdash\ &\langle \mathrm{TRUE}, \{\mathsf{f}^3(\mathsf{a}^3) \equiv \mathsf{a}^4,\ \mathsf{f}^4(\mathsf{a}^5) \equiv \mathsf{a}^6\}\rangle.
\end{aligned}$$

Now we test whether the indexed derivation is a *sound* derivation from AX' and we *identify* indexed function symbols with *common root* and *different indices* (only) if *necessary*. This must always succeed, because the original derivation is obtained if all indexed function symbol with common root are identified. Here the identifications $\mathsf{f}^2 = \mathsf{f}^3$, $\mathsf{a}^1 = \mathsf{a}^3$, $\mathsf{f}^1 = \mathsf{f}^4$, $\mathsf{a}^4 = \mathsf{a}^5$, and $\mathsf{a}^2 = \mathsf{a}^6$ are required for obtaining a sound derivation from AX'. Note that all indexed function symbols in the derivation are identified, except f^1, f^2 and a^1, a^2, a^4 because an identification of these symbols is not necessary for a sound derivation.

Finally, the recognized identifications are propagated into the conjecture φ', the set of axioms AX', and the derivation, yielding, e.g., $\varphi'' = \mathsf{f}^1(\mathsf{f}^2(\mathsf{a}^1)) \equiv \mathsf{a}^2$, $AX'' = \{\mathsf{f}^2(\mathsf{a}^1) \equiv \mathsf{a}^4,\ \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2, \ldots\}$, and the derivation

$$\begin{aligned}
&\langle \mathsf{f}^1(\mathsf{f}^2(\mathsf{a}^1)) \equiv \mathsf{a}^2, \emptyset\rangle\\
\vdash^{\mathrm{a}}_{AX''}\ &\langle \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2, \{\mathsf{f}^2(\mathsf{a}^1) \equiv \mathsf{a}^4\}\rangle\\
\vdash^{\mathrm{a}}_{AX''}\ &\langle \mathsf{a}^2 \equiv \mathsf{a}^2, \{\mathsf{f}^2(\mathsf{a}^1) \equiv \mathsf{a}^4,\ \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2\}\rangle\\
\vdash^{\mathrm{a}}_{AX''}\ &\langle \mathrm{TRUE}, \{\mathsf{f}^2(\mathsf{a}^1) \equiv \mathsf{a}^4,\ \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2\}\rangle
\end{aligned}$$

in the simple analysis calculus. From this derivation the proof shell PS^* from Example 6.2 is obtained by generalizing φ'' and the proof catch $\{\mathsf{f}^2(\mathsf{a}^1) \equiv \mathsf{a}^4, \mathsf{f}^1(\mathsf{a}^4) \equiv \mathsf{a}^2\}$.

Subsequently we shall develop the calculus of *refined proof analysis* yielding proof shells with significantly increased reusability, because *more general* schematic conjectures and catches are obtained as compared to the simple analysis approach. Proof shells based on the results of refined analysis *apply more often* and yield *weaker prove obligations* as the above examples illustrate.

The calculus of *refined proof analysis* emerges from the simple analysis calculus by the derivation of

(1) an *indexed* conjecture from a set of *indexed* axioms,
(2) an accumulator holding the *indexed* axioms used in the derivation, and
(3) a *collision set* for the bookkeeping of the *identifications* of the indexed symbols required for a sound derivation.

Definition 6.3 (*Indexed function symbols, indexed language*).

$$\Sigma_n^{\mathbb{N}} := \{f^i \mid f \in \Sigma_n,\ i \in \mathbb{N}\}$$

is the set of *indexed function symbols* for the object signature Σ. $\mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V})$ is called the set of *indexed terms* and $\mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ is the set of *indexed formulas*.

$$\mathit{index}: \mathcal{T}(\Sigma, \mathcal{V}) \cup \mathcal{F}(\Sigma, \mathcal{V}) \to \mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V}) \cup \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$$

is a mapping which supplies all occurrences of all function symbols in a term respectively formula with fresh indices. We demand *index* to yield indices in ascending order starting with 1.

$$\mathit{unindex}: \mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V}) \cup \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V}) \to \mathcal{T}(\Sigma, \mathcal{V}) \cup \mathcal{F}(\Sigma, \mathcal{V})$$

is the inverse mapping which removes the indices of all indexed function symbols in an indexed term respectively formula.

Strictly speaking, *index* is an operation with a side effect since we demand that indices obtained by *index* have been "never used before". Thus, e.g., $\mathit{index}(\mathsf{g}(\mathsf{f}(\mathsf{g}(x), y))) = \mathsf{g}^1(\mathsf{f}^1(\mathsf{g}^2(x), y))$ may result when indexing is started, but later $\mathit{index}(\mathsf{g}(\mathsf{f}(\mathsf{g}(x), y))) = \mathsf{g}^3(\mathsf{f}^2(\mathsf{g}^4(x), y))$ is obtained for the same input term. Consequently $\mathit{unindex}(\mathit{index}(t)) = t$ for each $t \in \mathcal{T}(\Sigma, \mathcal{V})$, but $\mathit{index}(\mathit{unindex}(t)) \neq t$ for each $t \in \mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V})$.

Definition 6.4 (*Index collision set, equivalence* $\sim_K$). A pair $\langle f^i, f^j \rangle \in \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$ is called an *index collision* and a set $K \subseteq \{\langle f^i, f^j \rangle \mid f \in \Sigma,\ i, j \in \mathbb{N}\} \subseteq \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$ is called an *index collision set*. We let $\sim_K$ denote the reflexive, transitive and symmetrical closure of K, i.e., $\sim_K$ is an equivalence relation on $\Sigma^{\mathbb{N}}$. $\lceil f^i \rceil_K := f^k$ is the *K-representative* of $f^i \in \Sigma^{\mathbb{N}}$ iff $k := \min\{j \mid f^i \sim_K f^j\}$.

Index collision sets are used to represent identifications for the different occurrences of function symbols, where "earlier" occurrences have smaller indices. Thus the K-representative of an indexed function symbol f^i is the first occurrence of f which is identified with f^i to satisfy the identifications represented by K.

Definition 6.5 (*Congruence* $\approx_K$, *minimal collision set* $=_K$). For an index collision set K, the equivalence relation $\sim_K$ on $\Sigma^{\mathbb{N}}$ is extended to a *congruence relation* $\approx_K$ on $\mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V})$, i.e., an equivalence relation such that $x \approx_K y$ iff $x = y$ for $x, y \in \mathcal{V}$ and

$$f^i(t_1, \ldots, t_n) \approx_K f^j(s_1, \ldots, s_n)$$

iff $f^i \sim_K f^j$ and $t_1 \approx_K s_1, \ldots, t_n \approx_K s_n$. We write $t =_K s$ iff $t \approx_K s$ for a *minimal* index collision set K, i.e., $t \approx_L s$ implies $\sim_K \subseteq \sim_L$ for each collision set L.

Note that only those indexed terms are *K-congruent* from which *identical* terms are obtained if the indices are removed. For instance $\mathsf{f}^1(\mathsf{f}^2(x)) \approx_K \mathsf{f}^3(\mathsf{f}^2(x))$ for $K := \{\langle \mathsf{f}^1, \mathsf{f}^3\rangle\}$, but $\mathsf{f}^1(\mathsf{f}^2(x)) \not\approx_{K'} \mathsf{f}^3(\mathsf{g}^1(x))$ for each index collision set K'.

Matching is extended to indexed terms by ignoring the indices when the clash test is performed.

Definition 6.6 (*Matching of indexed terms*). A substitution θ is an *indexed matcher* of $t, s \in \mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V})$ iff $\theta(t) \approx_K s$ for some collision set K. θ is a *minimal* indexed matcher of $t, s \in \mathcal{T}(\Sigma^{\mathbb{N}}, \mathcal{V})$ iff there is a collision set K such that $\theta(t) \approx_K s$ and $\sim_K \subseteq \sim_L$ for each indexed matcher λ with $\lambda(t) \approx_L s$ for a collision set L.

This means that, e.g., $\{x/a^1\}$ is an indexed matcher of $f^1(x)$ and $f^2(a^1)$ whereas $f^1(x)$ and $g^1(a^1)$ are non-matchable indexed terms. Further, e.g., $\{x/a^1\}$ or $\{x/a^2\}$ are minimal indexed matchers of $h^1(x, x)$ and $h^2(a^1, a^2)$, while $\{x/a^3\}$ is a non-minimal indexed matcher. Minimal indexed matchers never introduce indices which do not already occur in the matched terms.

The simple analysis calculus from Section 3.1 now is modified to incorporate the bookkeeping of the indices. The modified calculus operates on triples $\langle \forall x^*\ H \to C, A, K\rangle$, where all non-variable terms in H, C and A are indexed and an additional component, viz. the index collision set K, keeps track of the function symbols from $\Sigma^{\mathbb{N}}$ which have been identified in a proof.

The conjecture-sequent $\varphi = \forall x^*\ H \to C$ whose proof is going to be analyzed must be indexed *before* the derivation is performed, i.e., we start with the sequent $\mathit{index}(\varphi)$. The axioms of AX are freshly indexed before application, such that no indexed function symbol has more than one occurrence in the indexed accumulator and the indexed conjecture.

Definition 6.7 (*Refined analysis calculus*). The *refined analysis calculus* consists of the following inference rules operating on triples $\langle \varphi, A, K\rangle$ of an indexed sequent φ, an indexed accumulator A with $\mathit{unindex}(A) \subseteq AX$, and an index collision set $K \subseteq \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$, where AX is a set of axioms. θ is a minimal indexed matcher and p is a position in C:

- *Reflexivity*

$$\frac{\langle \forall x^*\ H \to s \equiv t, A, K\rangle}{\langle \mathrm{TRUE}, A, K \cup K'\rangle} \quad \text{if } s =_{K'} t.$$

- *AX-replacement*

$$\frac{\langle \forall x^*\ H \to C, A, K\rangle}{\langle \forall x^* H \to C[p \leftarrow \theta(r')], A \cup \{\forall u^*\ l' \equiv r'\}, K \cup K'\rangle}$$

  if $\forall u^*\ l \equiv r \in AX,\ l \notin u^*,\ \ l' := \mathit{index}(l),\ r' := \mathit{index}(r),\ C|_p =_{K'} \theta(l'),\ \mathcal{V}(r) \subseteq \mathcal{V}(l)$, and $\mathrm{dom}(\theta) = \mathcal{V}(l)$.

- *HYP-replacement*

$$\frac{\langle \forall x^*\ H \to C, A, K\rangle}{\langle \forall x^*\ H \to C[p \leftarrow \theta(r)], A, K \cup K'\rangle}$$

  if $\forall u^*\ l \equiv r \in H,\ l \notin u^*,\ C|_p =_{K'} \theta(l),\ (\mathcal{V}(r) \setminus \mathcal{V}(l)) \cap u^* = \emptyset$, and $\mathrm{dom}(\theta) = \mathcal{V}(l) \cap u^*$.

A sequence $\langle\langle\varphi_1, A_1, K_1\rangle, \ldots, \langle\varphi_n, A_n, K_n\rangle\rangle$ of triples of indexed sequents φ_i, indexed accumulators A_i with $\mathit{unindex}(A_i) \subseteq AX$, and index collision sets $K_i \subseteq \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$ is a *derivation* in the refined analysis calculus from a set AX of equational axioms iff for each $i \in \{1, \ldots, n-1\}$, $\langle\varphi_{i+1}, A_{i+1}, K_{i+1}\rangle$ results from applying one of the rules to $\langle\varphi_i, A_i, K_i\rangle$. Derivability is denoted by $\langle\varphi_1, A_1, K_1\rangle \vdash^{\mathrm{ak}}_{AX} \langle\varphi_n, A_n, K_n\rangle$, where AX may be omitted if appropriate.

In the refined analysis calculus both replacement rules not only differ in updating the accumulator component (as they already do in the simple analysis approach), but also differ in the treatment of the function indices: AX-replacement generates a freshly indexed variant of an axiom before application whereas *HYP*-replacement applies an already indexed hypothesis without index modifications. The indexed function symbols which have to be identified in a replacement step are recorded by both rules in the collision component.

Example 6.8 (*Refined proof analysis*). We resume Example 3.4. Consider the equation $E := [\forall x\ \mathsf{f}(\mathsf{c}, x) \equiv \mathsf{h}(x)]$ and the indexed expression

$$\big\langle H \to \mathsf{g}^2(\mathsf{f}^2(\mathsf{c}^2, \mathsf{h}^2(y))) \equiv \ldots, \{\ldots\}, \{\ldots\}\big\rangle.$$

If (an indexed version of) E is in H, e.g., $[\forall x\ \mathsf{f}^1(\mathsf{c}^1, x) \equiv \mathsf{h}^1(x)] \in H$, then

$$\big\langle H \to \mathsf{g}^2(\mathsf{h}^1(\mathsf{h}^2(y))) \equiv \ldots,\ \{\ldots\},\ \{\langle\mathsf{f}^2, \mathsf{f}^1\rangle, \langle\mathsf{c}^2, \mathsf{c}^1\rangle,\ \ldots\}\big\rangle$$

is obtained by *HYP*-replacement. But if $E \in AX$, then $\mathit{index}(E) := [\forall x\ \mathsf{f}^3(\mathsf{c}^3, x) \equiv \mathsf{h}^3(x)]$ is used by AX-replacement yielding the indexed expression

$$\big\langle H \to \mathsf{g}^2(\mathsf{h}^3(\mathsf{h}^2(y))) \equiv \ldots,\ \{\forall x\ \mathsf{f}^3(\mathsf{c}^3, x) \equiv \mathsf{h}^3(x), \ldots\},\ \{\langle\mathsf{f}^2, \mathsf{f}^3\rangle, \langle\mathsf{c}^2, \mathsf{c}^3\rangle, \ldots\}\big\rangle.$$

Note that we demand "$\ldots =_{K'} \ldots$" and use *minimal* indexed matchers in the definition of the inference rules. This entails that only function symbols are identified which *must* be identified and therefore guarantees that a *most general* (schematic) catch will be obtained subsequently. Otherwise, e.g., the pair $\langle\mathsf{h}^1, \mathsf{h}^2\rangle$ could also be inserted into the index collision sets of Example 6.8 yielding an identification which is not required by the inference steps.

For proving a formula φ, we try to establish $\langle \mathit{index}(\varphi), \emptyset, \emptyset\rangle \vdash^{\mathrm{ak}}_{AX} \langle \mathrm{TRUE}, A, K\rangle$ in the refined calculus. The collision set K then contains pairs $\langle f^i, f^j\rangle$ of indexed function symbols which have been identified in the proof. This information must be propagated into the accumulator A and into the conjecture $\mathit{index}(\varphi)$ when building the proof catch, because the proof of $\mathit{index}(\varphi)$ is based on the assumption that f^i and f^j denote *identical* functions: Either

(i) f^i and f^j are identified *syntactically* by replacing f^i with f^j (or vice versa) in the equations of A and $\mathit{index}(\varphi)$, or

(ii) f^i and f^j are identified *semantically* by insertion of the *identification axiom* $[\forall x_1, \ldots, x_n\ f^i(x_1, \ldots, x_n) \equiv f^j(x_1, \ldots, x_n)]$ into A.

The kind of identification influences the generality of the proof shell and the effort which must be spent for reusing the proof:

Example 6.9 (*Syntactical and semantical identification*). Let $AX = \{\mathsf{a} \equiv \mathsf{b},\ \mathsf{a} \equiv \mathsf{c},\ \mathsf{f}(x, y) \equiv \mathsf{g}(x, y), \ldots\}$ and consider the conjecture $\varphi = \mathsf{f}(\mathsf{a}, \mathsf{a}) \equiv \mathsf{f}(\mathsf{b}, \mathsf{c})$. Then

$$\begin{aligned}
&\langle \mathsf{f}^1(\mathsf{a}^1, \mathsf{a}^2) \equiv \mathsf{f}^2(\mathsf{b}^1, \mathsf{c}^1), \emptyset, \emptyset\rangle\\
\vdash^{\mathrm{ak}}_{AX}\ &\langle \mathsf{f}^1(\mathsf{b}^2, \mathsf{a}^2) \equiv \mathsf{f}^2(\mathsf{b}^1, \mathsf{c}^1), \{\mathsf{a}^3 \equiv \mathsf{b}^2\}, \{\mathsf{a}^1 = \mathsf{a}^3\}\rangle\\
\vdash^{\mathrm{ak}}_{AX}\ &\langle \mathsf{f}^1(\mathsf{b}^2, \mathsf{c}^2) \equiv \mathsf{f}^2(\mathsf{b}^1, \mathsf{c}^1), \{\mathsf{a}^3 \equiv \mathsf{b}^2, \mathsf{a}^4 \equiv \mathsf{c}^2\}, \{\mathsf{a}^1 = \mathsf{a}^3, \mathsf{a}^2 = \mathsf{a}^4\}\rangle\\
\vdash^{\mathrm{ak}}_{AX}\ &\langle \mathrm{TRUE}, \{\mathsf{a}^3 \equiv \mathsf{b}^2, \mathsf{a}^4 \equiv \mathsf{c}^2\}, \{\mathsf{a}^1 = \mathsf{a}^3, \mathsf{a}^2 = \mathsf{a}^4, \mathsf{f}^1 = \mathsf{f}^2, \mathsf{b}^1 = \mathsf{b}^2, \mathsf{c}^1 = \mathsf{c}^2\}\rangle
\end{aligned}$$

and *syntactical identification* yields the proof catch $\{\mathsf{a}^1 \equiv \mathsf{b}^1, \mathsf{a}^2 \equiv \mathsf{c}^1\}$ for the conjecture $\mathsf{f}^1(\mathsf{a}^1, \mathsf{a}^2) \equiv \mathsf{f}^1(\mathsf{b}^1, \mathsf{c}^1)$. After generalization, the proof shell

$$PS_{syn} = \langle F^1(A^1, A^2) \equiv F^1(B^1, C^1),\ \{A^1 \equiv B^1, A^2 \equiv C^1\}\rangle$$

is obtained from the proof.[11] *Semantical identification* yields the proof catch $\{\mathsf{a}^3 \equiv \mathsf{b}^2, \mathsf{a}^4 \equiv \mathsf{c}^2, \mathsf{a}^1 \equiv \mathsf{a}^3, \mathsf{a}^2 \equiv \mathsf{a}^4, \mathsf{f}^1(x, y) \equiv \mathsf{f}^2(x, y), \mathsf{b}^1 \equiv \mathsf{b}^2, \mathsf{c}^1 \equiv \mathsf{c}^2\}$ for the conjecture $\mathsf{f}^1(\mathsf{a}^1, \mathsf{a}^2) \equiv \mathsf{f}^2(\mathsf{b}^1, \mathsf{c}^1)$ which is generalized to the proof shell

$$\begin{aligned}
PS_{sem} = \big\langle & F^1(A^1, A^2) \equiv F^2(B^1, C^1),\\
& \{A^1 \equiv B^1, A^2 \equiv C^1, A^3 \equiv B^2, A^4 \equiv C^2, A^1 \equiv A^3,\\
& A^2 \equiv A^4, F^1(u, v) \equiv F^2(u, v), B^1 \equiv B^2, C^1 \equiv C^2\}\big\rangle.
\end{aligned}$$

Obviously PS_{sem} is *more general* than PS_{syn} in the sense that PS_{sem} applies for a conjecture ψ and provable proof obligations are obtained from the catch of PS_{sem} whenever PS_{syn} applies for ψ such that provable proof obligations are obtained from the catch of PS_{syn}, but *not vice versa*.

For instance, PS_{sem} applies for the conjecture $\psi = \mathsf{f}(\mathsf{a}, \mathsf{a}) \equiv \mathsf{g}(\mathsf{b}, \mathsf{c})$ via the matcher $\pi = \{F^1/\mathsf{f}, A^1/\mathsf{a}, A^2/\mathsf{a}, F^2/\mathsf{g}, B^1/\mathsf{b}, C^1/\mathsf{c}\}$, hence the partially instantiated catch $\pi(C_{sem})$ is computed as $\{\mathsf{a} \equiv \mathsf{b}, \mathsf{a} \equiv \mathsf{c}, A^3 \equiv B^2, A^4 \equiv C^2, \mathsf{a} \equiv A^3, \mathsf{a} \equiv A^4, \mathsf{f}(x, y) \equiv \mathsf{g}(x, y), \mathsf{b} \equiv B^2, \mathsf{c} \equiv C^2\}$. Since $\rho = \{A^3/\mathsf{a}, A^4/\mathsf{a}, B^2/\mathsf{b}, C^2/\mathsf{c}\}$ is a solution for the free function variables in C_{sem}, i.e.,

$$AX \models \rho(\pi(C_{sem})) = \{\mathsf{a} \equiv \mathsf{b}, \mathsf{a} \equiv \mathsf{c}, \mathsf{a} \equiv \mathsf{a}, \mathsf{f}(x, y) \equiv \mathsf{g}(x, y), \mathsf{b} \equiv \mathsf{b}, \mathsf{c} \equiv \mathsf{c}\},$$

ψ is proved by reuse based on the proof shell PS_{sem}. However, PS_{syn} does not apply for ψ because the function variable F^1 cannot be replaced both by f and g, and consequently a reuse based on PS_{syn} fails for ψ.

Example 6.9 illustrates that semantical identification yields *more general proof shells*, however for the price of *larger* schematic catches with *more function variables* as

[11] We use *indexed function variables* in examples only for illustrational purposes and the sake of the presentation. This means that, e.g., A^1 and A^2 are *different* function variables sharing no common property as, e.g., A^1 and B^1 have no property in common.

compared to syntactical identification. Hence semantical identification increases the effort of reuse as *more* bound function variables must be *matched*, *more* free function variables must be *solved* and *more* proof obligation must be *verified*. Correspondingly, syntactical identification minimizes the reuse effort, however for the price of a restricted applicability of the proof shell. Consequently, some compromise must be made between both extremes, and we propose a quite simple criterion to decide between both forms of identification which combines the advantages of both alternatives and has proved sufficient: *Bound* function symbols, i.e., function symbols stemming from the indexed conjecture, are identified *semantically* so that the schematic conjecture of a proof shell is as general as possible thus increasing applicability. *Free* function symbols, i.e., function symbols occurring only in the proof catch, are identified *syntactically* so that the number of free function variables and the size of the schematic catch is minimized thus decreasing the reuse effort.

Definition 6.10 (*Identification heuristic*). Let $\langle \varphi, \emptyset, \emptyset \rangle \vdash^{\mathrm{ak}}_{AX} \langle \mathrm{TRUE}, A, K \rangle$. Then the collision set K defines a mapping

$$\mathrm{id}_K : 2^{\mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})} \to 2^{\mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})}$$

such that each formula $\psi' \in \mathrm{id}_K(A)$ is obtained from $\psi \in A$ by replacing each *free* function symbol f^i in ψ with its K-representative $\lceil f^i \rceil_K$, cf. Definition 6.4. We call $\mathrm{id}_K(A)$ the *syntactic* catch of the proof of φ.

The collision set K also defines a set of indexed formulas $\mathrm{ID}_K \subseteq \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ such that $\forall x_1, \ldots, x_n \lceil f^i \rceil_K(x_1, \ldots, x_n) \equiv f^i(x_1, \ldots, x_n) \in \mathrm{ID}_K$ iff f^i is a *bound* function symbol with $f^i \neq \lceil f^i \rceil_K$. ID_K is called the *semantic catch* of the proof of φ and each member of ID_K is an *identification axiom*.

Since we assign indices in *ascending* order starting with the conjecture, a free function symbol f^i is always replaced by a bound function symbol if the $\sim_K$-equivalence class of f^i contains one bound function symbol at least. Thus the function symbols from the conjecture are propagated into the proof catch as far as possible (and necessary). The union of the syntactic and the semantic catch now form the catch of the indexed conjecture's proof, which then is generalized to a proof shell:

Lemma 6.11 (Refined versus simple analysis). *Let $AX \subset \mathcal{F}(\Sigma, \mathcal{V})$ and $A \subset \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ be sets of universally closed equations, let $K \subset \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$ be a collision set and let φ be a sequent in $\mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ such that $\langle \varphi, \emptyset, \emptyset \rangle \vdash^{\mathrm{ak}}_{AX} \langle \mathrm{TRUE}, A, K \rangle$. Then*

$$\langle \varphi, \emptyset \rangle \vdash^{\mathrm{a}}_{id_K(A) \cup ID_K} \langle \mathrm{TRUE}, id_K(A) \cup ID_K \rangle.$$

Proof. Transforming the derivation from the refined calculus into a derivation in the simple analysis calculus succeeds because "index disagreements" can be removed: Each step of the derivation $\vdash^{\mathrm{ak}}_{AX}$ is transformed into a sequence of replacement steps in $\vdash^{\mathrm{a}}_{\mathrm{ID}_K}$ with the AX-replacement-rule (where identification axioms from ID_K are used for adapting colliding indices of bound function symbols if necessary) followed by one step in $\vdash^{\mathrm{a}}_{\mathrm{id}_K(A)}$ with the same rule which has been used in $\vdash^{\mathrm{ak}}_{AX}$. □

Theorem 6.12 (Soundness of $\vdash^{\mathrm{ak}}_{AX}$). *Let $AX \subset \mathcal{F}(\Sigma, \mathcal{V})$ and $A \subset \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ be sets of universally closed equations, let $K \subset \Sigma^{\mathbb{N}} \times \Sigma^{\mathbb{N}}$ be a collision set and let φ be a sequent in $\mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$ such that $\langle \varphi, \emptyset, \emptyset\rangle \vdash^{\mathrm{ak}}_{AX} \langle \mathrm{TRUE}, A, K\rangle$. Then*

(i) *$unindex(A) \subset AX$, $|A| < \infty$, and $\langle \varphi, \emptyset, \emptyset\rangle \vdash^{\mathrm{ak}}_{unindex(A)} \langle \mathrm{TRUE}, A, K\rangle$.*
(ii) *$id_K(A) \cup ID_K \models \varphi$, and*
(iii) *$AX \models unindex(\varphi)$.*

Proof. (i) $unindex(A) \subset AX$ is obvious from the definition of A by the rules of the calculus and $|A| < \infty$ since each derivation is finite. Finally each derivation from AX also is a derivation from $unindex(A)$ since no axioms from $AX \backslash unindex(A)$ are used in the derivation.

(ii) We find

$$\langle \varphi, \emptyset\rangle \vdash^{\mathrm{a}}_{id_K(A) \cup ID_K} \langle \mathrm{TRUE}, id_K(A) \cup ID_K\rangle$$

by Lemma 6.11 and then the statement follows by Theorem 3.5(ii).

(iii) Since $id_K(A) \cup ID_K \models \varphi$ by (ii), $unindex(id_K(A)) \cup unindex(ID_K) \models unindex(\varphi)$. With $unindex(id_K(A)) = unindex(A)$ and $\models unindex(ID_K)$, we obtain $unindex(A) \models unindex(\varphi)$. With $unindex(A) \subset AX$, cf. (i), the statement follows by the monotonicity of semantical entailment. □

Corollary 6.13 (Proof shells from refined analysis). *Let $A \cup \{\varphi\} \subset \mathcal{F}(\Sigma^{\mathbb{N}}, \mathcal{V})$, such that*

$$\langle \varphi, \emptyset, \emptyset\rangle \vdash^{\mathrm{ak}}_{unindex(A)} \langle \mathrm{TRUE}, A, K\rangle$$

and let γ be a generalization function such that $\gamma(A \cup \{\varphi\}) \subset \mathcal{F}(\Omega, \mathcal{U})$. Then $\langle \gamma(\varphi), \gamma(id_K(A) \cup ID_K)\rangle$ is a proof shell.

Proof. $\langle \varphi, \emptyset\rangle \vdash^{\mathrm{a}}_{id_K(A) \cup ID_K} \langle \mathrm{TRUE}, id_K(A) \cup ID_K\rangle$ by Lemma 6.11 and then the statement follows by Theorem 3.9. □

Example 6.14 (*Identification heuristic*). We resume Example 6.9 and apply the identification heuristic to the conjecture $\mathsf{f}^1(\mathsf{a}^1, \mathsf{a}^2) \equiv \mathsf{f}^2(\mathsf{b}^1, \mathsf{c}^1)$, the catch $A = \{\mathsf{a}^3 \equiv \mathsf{b}^2, \mathsf{a}^4 \equiv \mathsf{c}^2\}$ and the collision set $K = \{\mathsf{a}^1 = \mathsf{a}^3, \mathsf{a}^2 = \mathsf{a}^4, \mathsf{f}^1 = \mathsf{f}^2, \mathsf{b}^1 = \mathsf{b}^2, \mathsf{c}^1 = \mathsf{c}^2\}$: Since all function symbols in A are free, syntactical identification yields the syntactic catch $\mathrm{id}_K(A) = \{\mathsf{a}^1 \equiv \mathsf{b}^1, \mathsf{a}^2 \equiv \mathsf{c}^1\}$, and the semantic catch ID_K is computed as $\{\mathsf{f}^1(x, y) \equiv \mathsf{f}^2(x, y)\}$, because f^1 and f^2 are bound and K-equivalent. From the union of both catches, the proof shell

$$\begin{aligned} PS_{heu} = \langle & F^1(A^1, A^2) \equiv F^2(B^1, C^1), \\ & \{A^1 \equiv B^1, A^2 \equiv C^1, F^1(u, v) \equiv F^2(u, v)\}\rangle \end{aligned}$$

is obtained, which is *more general* than PS_{syn} (but *less general* than PS_{sem}).

For instance, PS_{heu} applies for $\psi = \mathsf{f}(\mathsf{a}, \mathsf{a}) \equiv \mathsf{g}(\mathsf{b}, \mathsf{c})$ via the matcher $\pi = \{F^1/\mathsf{f}, A^1/\mathsf{a}, A^2/\mathsf{a}, F^2/\mathsf{g}, B^1/\mathsf{b}, C^1/\mathsf{c}\}$, hence the *totally* instantiated catch $\pi(C_{heu})$ is computed as $\{\mathsf{a} \equiv \mathsf{b}, \mathsf{a} \equiv \mathsf{c}, \mathsf{f}(x, y) \equiv \mathsf{g}(x, y)\} \subseteq AX$, and ψ is proved by reuse based on PS_{heu}. Obviously, the proof shell PS_{heu} applies more often than the proof shell PS_{syn}, e.g., for ψ, but requires no effort for solving free variables and yields less proof obligations than the proof shell PS_{sem}.

Definition 6.10 formulates a heuristic decision between syntactical and semantical identification which is *independent* of a reasoning domain, and modifications of this heuristic may be useful for a particular domain under consideration. Here, for instance, theorem proving by induction is considered and indexing of a step formula like φ_s, e.g., yields

$$(\forall z\ \mathsf{plus}^1(\mathsf{sum}^1(x), \mathsf{sum}^2(z)) \equiv \mathsf{sum}^3(\mathsf{app}^1(x, z))) \\ \rightarrow \mathsf{plus}^2(\mathsf{sum}^4(\mathsf{add}^1(n, x)), \mathsf{sum}^5(y)) \equiv \mathsf{sum}^6(\mathsf{app}^2(\mathsf{add}^2(n, x), y)).$$

Since step formulas always are proved by modifying the induction conclusion such that the induction hypothesis becomes applicable, the function symbols of the induction conclusion are always identified with the corresponding function symbols of the induction hypothesis, and we obtain $\mathsf{plus}^1 \sim_K \mathsf{plus}^2$, $\mathsf{sum}^1 \sim_K \mathsf{sum}^4$, $\mathsf{sum}^2 \sim_K \mathsf{sum}^5$, $\mathsf{sum}^3 \sim_K \mathsf{sum}^6$, $\mathsf{app}^1 \sim_K \mathsf{app}^2$, and $\mathsf{add}^1 \sim_K \mathsf{add}^2$. All these function symbols are *bound* and therefore identified *semantically* as Definition 6.10 demands, yielding a proof shell *PS* with the schematic conjecture

$$(\forall w\ F^1(G^1(u), G^2(w)) \equiv G^3(H^1(u, w))) \\ \rightarrow F^2(G^4(D^1(m, x)), G^5(v)) \equiv G^6(H^2(D^2(m, u), v))$$

and the set I of identification axioms $\{F^1(u, v) \equiv F^2(u, v), G^1(u) \equiv G^4(u), G^2(u) \equiv G^5(u), G^3(u) \equiv G^6(u), H^1(u, v) \equiv H^2(u, v), D^1(u, v) \equiv D^2(u, v)\}$ as a subset of the semantic catch. Now if *PS* applies for some step formula $IH \rightarrow IC$, corresponding function variables of the schematic conjecture as, e.g., F^1, F^2 or G^1, G^4 are replaced by the *same* functional terms, since the function symbols in the induction conclusion *IC* correspond to the function symbols in the induction hypothesis *IH*. For example, *PS* applies for $\psi =$

$$(\forall z\ \mathsf{times}(\mathsf{prod}(x), \mathsf{prod}(z)) \equiv \mathsf{prod}(\mathsf{app}(x, z))) \\ \rightarrow \mathsf{times}(\mathsf{prod}(\mathsf{add}(n, x)), \mathsf{prod}(y)) \equiv \mathsf{prod}(\mathsf{app}(\mathsf{add}(n, x), y))$$

and all identification axioms of I are instantiated to *tautologies* like $\mathsf{times}(x, y) \equiv \mathsf{times}(x, y)$, $\mathsf{prod}(k) \equiv \mathsf{prod}(k)$, etc. This example reveals that the identification heuristic applied to *step formulas* yields a proof shell with many *redundant schematic formulas* in the catch and a schematic conjecture with *useless generality*. As an obvious remedy to this problem, we only supply the *induction conclusion* of a step formula with *unique* indices and use the *same* indices in the *induction hypothesis*. For instance, $\varphi_s' =$

$$(\forall z\ \mathsf{plus}^1(\mathsf{sum}^1(x), \mathsf{sum}^2(z)) \equiv \mathsf{sum}^3(\mathsf{app}^1(x, z))) \\ \rightarrow \mathsf{plus}^1(\mathsf{sum}^1(\mathsf{add}^1(n, x)), \mathsf{sum}^2(y)) \equiv \mathsf{sum}^3(\mathsf{app}^1(\mathsf{add}^1(n, x), y))$$

is used as the indexed version of φ_s. The corresponding schematic conjecture

$$(\forall w\ F^1(G^1(u), G^2(w)) \equiv G^3(H^1(u, w))) \\ \rightarrow F^1(G^1(D^1(m, x)), G^2(v)) \equiv G^3(H^1(D^1(m, u), v))$$

still applies for ψ, but redundant proof obligations now are avoided thus decreasing the costs of reuse.

Example 6.15 (*Proof shells from refined proof analysis*). We resume the proof of the step formula φ_s from Section 2 for the conjecture $\varphi = \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(x, y))$, but now with *refined proof analysis*. For the indexed step formula φ'_s from above, a derivation $\langle \varphi'_s, \emptyset, \emptyset \rangle \vdash^{\mathrm{ak}}_{AX} \langle \mathrm{TRUE}, A, K \rangle$ is obtained such that

$$\begin{array}{lll} \mathsf{plus}^1 \sim_K \mathsf{plus}^4 \sim_K \mathsf{plus}^7, & \mathsf{plus}^2 \sim_K \mathsf{plus}^5, & \mathsf{plus}^6 \sim_K \mathsf{plus}^3, \\ \mathsf{add}^1 \sim_K \mathsf{add}^2 \sim_K \mathsf{add}^3, & \mathsf{add}^4 \sim_K \mathsf{add}^5, & \mathsf{app}^1 \sim_K \mathsf{app}^2 \sim_K \mathsf{app}^3, \\ \mathsf{sum}^1 \sim_K \mathsf{sum}^4 \sim_K \mathsf{sum}^5, & \mathsf{sum}^3 \sim_K \mathsf{sum}^6 \sim_K \mathsf{sum}^7. & \end{array}$$

The indexed accumulator A contains indexed copies of the applied axioms and is obtained as

$$A := \left\{ \begin{array}{ll} \mathit{index}(\mathsf{sum\text{-}2}) & \mathsf{sum}^4(\mathsf{add}^2(n, x)) \equiv \mathsf{plus}^2(n, \mathsf{sum}^5(x)) \\ \mathit{index}(\mathsf{app\text{-}2}) & \mathsf{app}^2(\mathsf{add}^3(n, x), y) \equiv \mathsf{add}^4(n, \mathsf{app}^3(x, y)) \\ \mathit{index}(\mathsf{sum\text{-}2}) & \mathsf{sum}^6(\mathsf{add}^5(n, x)) \equiv \mathsf{plus}^3(n, \mathsf{sum}^7(x)) \\ \mathit{index}(\mathsf{lem\text{-}1}) & \mathsf{plus}^4(\mathsf{plus}^5(x, y), z) \equiv \mathsf{plus}^6(x, \mathsf{plus}^7(y, z)) \end{array} \right\}.$$

Here $\mathrm{ID}_K = \emptyset$ since there are no collisions between *bound* function symbols, and the syntactical catch is computed as

$$c := \mathrm{id}_K(A) = \left\{ \begin{array}{ll} (13) & \mathsf{sum}^1(\mathsf{add}^1(n, x)) \equiv \mathsf{plus}^2(n, \mathsf{sum}^1(x)) \\ (14) & \mathsf{app}^1(\mathsf{add}^1(n, x), y) \equiv \mathsf{add}^4(n, \mathsf{app}^1(x, y)) \\ (15) & \mathsf{sum}^3(\mathsf{add}^4(n, x)) \equiv \mathsf{plus}^3(n, \mathsf{sum}^3(x)) \\ (16) & \mathsf{plus}^1(\mathsf{plus}^2(x, y), z) \equiv \mathsf{plus}^3(x, \mathsf{plus}^1(y, z)) \end{array} \right\}.$$

Now the equations in the indexed conjecture φ'_s and in the indexed catch c are generalized by $\gamma := \{\mathsf{plus}^1/F^1, \mathsf{plus}^2/F^2, \mathsf{plus}^3/F^3, \ \mathsf{sum}^1/G^1, \ \mathsf{sum}^2/G^2, \mathsf{sum}^3/G^3, \ \mathsf{app}^1/H^1, \mathsf{add}^1/D^1, \ \mathsf{add}^4/D^4, \ \ldots\}$,[12] and we obtain the proof shell $PS_1 = \langle \Phi, C \rangle$ of Fig. 4.

We illustrate the advantages of refined proof analysis with examples from the induction domain by comparing attempts of reusing proofs with the simple analysis approach and with the refined approach:

Example 6.16 (*Proof shells from simple and refined analysis*).

(i) For $\psi_3 := \mathsf{plus}(x, 0) \equiv x$, the step formula $\psi_{3s} := \mathsf{plus}(x, 0) \equiv x \to \mathsf{plus}(\mathsf{s}(x), 0) \equiv \mathsf{s}(x)$ is computed, but the proof shell PS_s from Fig. 1 does not apply since the function variable G from the schematic conjecture cannot be replaced both by w_1 and 0. Hence ψ_{3s} cannot be proved by reuse based on PS_s. But PS_1 from Fig. 4 applies via the matcher $\pi_3 := \{F^1/\mathsf{plus}, \ G^{1,3}/w_1, G^2/0, \ H^1/w_1, \ D^1/\mathsf{s}(w_2)\}$ [13]

[12] Here we assume appropriate generalizations of the object variables. Since the equations in the catch are (implicitly) universally quantified, e.g., the occurrences of n in (13) and (14) from Example 6.15 denote *different variables* which must also be generalized differently.

[13] We use expressions like "$G^{1,3}/w_1$" as a shorthand for "$G^1/w_1, \ G^3/w_1$".

$$\Phi := (\forall u\ F^1(G^1(x), G^2(u)) \equiv G^3(H^1(x,u))) \rightarrow F^1(G^1(D^1(n,x)), G^2(y)) \equiv G^3(H^1(D^1(n,x),y))$$

$$C := \left\{ \begin{array}{ll} (17) & G^1(D^1(n,x)) \equiv F^2(n, G^1(x)) \\ (18) & H^1(D^1(n,x), y) \equiv D^4(n, H^1(x,y)) \\ (19) & G^3(D^4(n,x)) \equiv F^3(n, G^3(x)) \\ (20) & F^1(F^2(x,y),z) \equiv F^3(x, F^1(y,z)) \end{array} \right\}$$

Fig. 4. The proof shell PS_1 obtained from the proof of φ_s.

and an appropriate first-order variable renaming ν_3. With the solution substitution $\rho_3 := \{D^4/\mathsf{s}(w_2),\ F^{2,3}/\mathsf{s}(w_2)\}$ for the free function variables the totally instantiated catch $\rho_3(\pi_3(\nu_3(C)))$ is obtained as

$$\rho_3(\pi_3(\nu_3(C))) := \left\{ \begin{array}{ll} (21) & \mathsf{s}(x) \equiv \mathsf{s}(x) \\ (22) & \mathsf{s}(x) \equiv \mathsf{s}(x) \\ (23) & \mathsf{s}(x) \equiv \mathsf{s}(x) \\ (24) & \mathsf{plus}(\mathsf{s}(y), z) \equiv \mathsf{s}(\mathsf{plus}(y,z)) \end{array} \right\}.$$

Since $\{\pi_3, \rho_3, \nu_3\}$ is admissible for $C \cup \{\Phi\}$ and $AX \models \rho_3(\pi_3(\nu_3(C)))$, ψ_{3s} is proved by reuse based on PS_1, cf. Theorem 4.3. Here the *different* instantiations $G^{1,3}/w_1$ versus $G^2/0$ are essential for the success.

(ii) Let $\psi_4 := \mathsf{plus}(\mathsf{len}(x), \mathsf{len}(y)) \equiv \mathsf{len}(\mathsf{app}(x,y))$ where the function len is given by the defining equations

$$\begin{array}{ll} \text{(len-1)} & \mathsf{len}(\mathsf{empty}) \equiv 0, \\ \text{(len-2)} & \mathsf{len}(\mathsf{add}(n,x)) \equiv \mathsf{s}(\mathsf{len}(x)). \end{array}$$

The step formula

$$\psi_{4s} := \big(\forall u\ \mathsf{plus}(\mathsf{len}(x), \mathsf{len}(u)) \equiv \mathsf{len}(\mathsf{app}(x,u))\big) \rightarrow \mathsf{plus}(\mathsf{len}(\mathsf{add}(n,x)), \mathsf{len}(y)) \equiv \mathsf{len}(\mathsf{app}(\mathsf{add}(n,x), y)),$$

is computed for ψ_4 and PS_s from Fig. 1 applies via the matcher $\pi_4 := \{F/\mathsf{plus}, G/\mathsf{len}, H/\mathsf{app}, D/\mathsf{add}\}$ and an appropriate first-order variable renaming ν_4. Hence we instantiate the schematic catch C_s correspondingly and obtain the proof obligations

$$\pi_4(\nu_4(C)) := \left\{ \begin{array}{ll} (25) & \mathsf{len}(\mathsf{add}(n,x)) \equiv \mathsf{plus}(n, \mathsf{len}(x)) \\ (26) & \mathsf{app}(\mathsf{add}(n,x), y) \equiv \mathsf{add}(n, \mathsf{app}(x,y)) \\ (27) & \mathsf{plus}(\mathsf{plus}(x,y), z) \equiv \mathsf{plus}(x, \mathsf{plus}(y,z)) \end{array} \right\}.$$

But statement (25) obviously does not hold, and therefore a reuse based on PS_s fails for ψ_{4s}. However, also PS_1 from Fig. 4 applies for ψ_{4s} via the matcher $\pi_4' := \{F^1/\mathsf{plus},\ G^{1,2,3}/\mathsf{len},\ H^1/\mathsf{app},\ D^1/\mathsf{add}\}$ and an appropriate first-order variable

renaming ν_4'. With the solution substitution $\rho_4' := \{D^4/\mathsf{add},\ F^{2,3}/\mathsf{s}(w_2)\}$ for the free function variables the totally instantiated catch $\rho_4'(\pi_4'(\nu_4'(C)))$ is obtained as

$$\rho_4'(\pi_4'(\nu_4'(C))) := \left\{ \begin{array}{ll} (28) & \mathsf{len}(\mathsf{add}(n,x)) \equiv \mathsf{s}(\mathsf{len}(x)) \\ (29) & \mathsf{app}(\mathsf{add}(n,x),y) \equiv \mathsf{add}(n,\mathsf{app}(x,y)) \\ (30) & \mathsf{len}(\mathsf{add}(n,x)) \equiv \mathsf{s}(\mathsf{len}(x)) \\ (31) & \mathsf{plus}(\mathsf{s}(y),z) \equiv \mathsf{s}(\mathsf{plus}(y,z)) \end{array} \right\}.$$

Since $\{\pi_4', \rho_4', \nu_4'\}$ is admissible for $C \cup \{\Phi\}$ and $AX \models \rho_4'(\pi_4'(\nu_4'(C)))$, ψ_{4s} is proved by reuse based on PS_1, cf. Theorem 4.3. Note that the *different* instantiations F^1/plus versus $F^{2,3}/\mathsf{s}(w_2)$ are essential for the success, and this is why the reuse attempt with the proof shell PS_s fails.

(iii) Let $\psi_5 := \mathsf{minus}(\mathsf{plus}(\mathsf{dhalf}(x), \mathsf{uhalf}(x)), x) \equiv 0$ where the functions minus, dhalf and uhalf are given by the defining equations

$$\begin{array}{ll} \text{(minus-1)} & \mathsf{minus}(0,y) \equiv 0, \\ \text{(minus-2)} & \mathsf{minus}(\mathsf{s}(x),0) \equiv \mathsf{s}(x), \\ \text{(minus-3)} & \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \equiv \mathsf{minus}(x,y), \\ \text{(dhalf-1)} & \mathsf{dhalf}(0) \equiv 0, \\ \text{(dhalf-2)} & \mathsf{dhalf}(\mathsf{s}(x)) \equiv \mathsf{uhalf}(x), \\ \text{(uhalf-1)} & \mathsf{uhalf}(0) \equiv 0, \\ \text{(uhalf-2)} & \mathsf{uhalf}(\mathsf{s}(x)) \equiv \mathsf{s}(\mathsf{dhalf}(x)). \end{array}$$

The step formula

$$\begin{array}{l} \psi_{5s} := \mathsf{minus}(\mathsf{plus}(\mathsf{dhalf}(x), \mathsf{uhalf}(x)), x) \equiv 0 \\ \qquad \rightarrow \mathsf{minus}(\mathsf{plus}(\mathsf{dhalf}(\mathsf{s}(x)), \mathsf{uhalf}(\mathsf{s}(x))), \mathsf{s}(x)) \equiv 0, \end{array}$$

is computed for ψ_5 and using the lemma

$$\text{(lem-2)} \quad \mathsf{plus}(x,y) \equiv \mathsf{plus}(y,x),$$

the proof shell PS_{5s} from Fig. 5 is obtained from the proof of ψ_{5s} by simple analysis

$$\begin{array}{l} \Phi_{5S} := \ F(K(H(x),G(x)),x) \equiv A \\ \qquad\qquad \rightarrow F(K(H(D(x)),G(D(x))),D(x)) \equiv A \end{array}$$

$$C_{5S} := \left\{ \begin{array}{ll} (32) & H(D(x)) \equiv G(x) \\ (33) & G(D(x)) \equiv D(H(x)) \\ (34) & K(x,y) \equiv K(y,x) \\ (35) & K(D(x),y) \equiv D(K(x,y)) \\ (36) & F(D(x),D(y)) \equiv F(x,y) \end{array} \right\}.$$

Fig. 5. The proof shell PS_{5s} obtained from the proof of ψ_{5s} (simple analysis).

$$\Phi_{5S'} := \; F(K(H(x), G(x)), x) \equiv A \\ \rightarrow F(K(H(D(x)), G(D(x))), D(x)) \equiv A$$

$$C_{5S'} := \left\{ \begin{array}{ll} (37) & H(D(x)) \equiv G(x) \\ (38) & G(D(x)) \equiv D^4(H(x)) \\ (39) & K(x, y) \equiv K^3(y, x) \\ (40) & K^3(D^4(x), y) \equiv D^6(K(x, y)) \\ (41) & F(D^6(x), D(y)) \equiv F(x, y) \end{array} \right\}.$$

Fig. 6. The proof shell $PS_{5s'}$ obtained from the proof of $\psi_{5s'}$ (refined analysis).

Now let $\psi_6 := \mathsf{or}(\mathsf{even}(x), \mathsf{odd}(x)) \equiv \mathsf{true}$ where the functions or, even and odd are given by the defining equations

$$\begin{array}{ll} \text{(or-1)} & \mathsf{or}(\mathsf{true}, y) \equiv \mathsf{true}, \\ \text{(or-2)} & \mathsf{or}(\mathsf{false}, y) \equiv y, \\ \text{(even-1)} & \mathsf{even}(0) \equiv \mathsf{true}, \\ \text{(even-2)} & \mathsf{even}(\mathsf{s}(x)) \equiv \mathsf{odd}(x), \\ \text{(odd-1)} & \mathsf{odd}(0) \equiv \mathsf{false}, \\ \text{(odd-2)} & \mathsf{odd}(\mathsf{s}(x)) \equiv \mathsf{even}(x). \end{array}$$

The step formula

$$\psi_{6s} := \mathsf{or}(\mathsf{even}(x), \mathsf{odd}(x)) \equiv \mathsf{true} \\ \rightarrow \mathsf{or}(\mathsf{even}(\mathsf{s}(x)), \mathsf{odd}(\mathsf{s}(x))) \equiv \mathsf{true}$$

is computed for ψ_6 and the proof shell PS_{5s} from Fig. 5 applies via the matcher $\pi_5 := \{F/w_1, K/\mathsf{or}, H/\mathsf{even}, G/\mathsf{odd}, D/\mathsf{s}, A/\mathsf{true}\}$ and an appropriate first-order variable renaming ν_5. However, $\{\pi_5, \nu_5\}$ is not admissible for $C_{5S} \cup \{\Phi_{5S}\}$, because π_5 demands a sort assignment, viz. $H : \mathsf{nat} \rightarrow \mathsf{bool}$ and $D : \mathsf{nat} \rightarrow \mathsf{nat}$, such that Eq. (33) $\ldots \equiv D(H(x))$ from C_{5S} is ill-sorted.

By refined analysis, the proof shell $PS_{5s'}$ from Fig. 6 is obtained. This proof shell also applies via the matcher π_5 and an appropriate first-order variable renaming ν_5 for the step formula ψ_{6s}, but here $\{\pi_5, \nu_5\}$ is admissible for $C_{5S'} \cup \{\Phi_{5S'}\}$, as the well-sortedness of π_5 does not spoil the well-sortedness of, e.g., Eq. (38) $\ldots \equiv D^4(H(x))$ from $C_{5S'}$.

With the solution substitution $\rho_5 := \{D^{4.6}/w_1, \; K^3/\mathsf{or}(w_2, w_1)\}$ for the free function variables the totally instantiated catch $\rho_5(\pi_5(\nu_5(C_{5S'})))$ is obtained as

$$\rho_5(\pi_5(\nu_5(C))) := \left\{ \begin{array}{ll} (42) & \mathsf{even}(\mathsf{s}(x)) \equiv \mathsf{odd}(x) \\ (43) & \mathsf{odd}(\mathsf{s}(x)) \equiv \mathsf{even}(x) \\ (44) & \mathsf{or}(x, y) \equiv \mathsf{or}(x, y) \\ (45) & \mathsf{or}(y, x) \equiv \mathsf{or}(x, y) \\ (46) & x \equiv x \end{array} \right\}.$$

Since $\{\pi_5, \rho_5, \nu_5\}$ is admissible for $C_{5S'} \cup \{\Phi_{5S'}\}$ and $AX \models \rho_5(\pi_5(\nu_5(C_{5S'} \setminus \{(45)\})))$, ψ_{6s} is proved by reuse based on $PS_{5S'}$, provided the *speculated lemma* (45) can be verified. Note that the introduction of the *free* function variables as, e.g., D^4 by refined analysis are essential for the success, and this is why the reuse attempt with the proof shell PS_{5s} fails.

7. Automated proof reuse

The developments of the preceding sections can be implemented in the following way: An (automated or interactive) first-order theorem prover computes proofs within the refined analysis calculus. From these analyzed proofs, proof shells according to Corollary 6.13 are obtained and collected in a *proof dictionary PD*, i.e., a collection of "proof ideas" organized as a finite set of proof shells. Now *before* the theorem prover is called for verifying a new conjecture ψ, the proof dictionary PD is searched for a proof shell $PS = \langle \Phi, C \rangle$ such that

(1) $\psi = \nu \circ \pi(\Phi)$,
(2) $\nu \circ \pi(C) \subset \mathcal{F}(\Sigma, \mathcal{V})$, and
(3) $AX \models_S \nu \circ \pi(C)$ for some schematic variable renaming ν and some schematic substitution π which are admissible for $C \cup \{\Phi\}$.

If successful, $AX \models_S \psi$ is verified by reuse, because $\nu \circ \pi(C) \models_S \psi$ by the Reuse Theorem 5.1.

We solve the problem of finding a proof shell PS and the substitutions ν and π for a given conjecture ψ and a given proof dictionary PD in three steps:

In the *retrieval step*, the proof dictionary PD is searched for a proof shell $PS = \langle \Phi, C \rangle$ such that $\psi = \nu_0 \circ \pi_0(\Phi)$ for some schematic variable renaming ν_0 and some schematic substitution π_0 which are admissible for $C \cup \{\Phi\}$. The composed substitution $\nu_0 \circ \pi_0$ is called a *retrieval matcher* of Φ and ψ and is applied to the schematic catch C yielding the *partially instantiated catch* $C_0 := \nu_0 \circ \pi_0(C)$.

If $C_0 \not\subset \mathcal{F}(\Sigma, \mathcal{V})$, some $\Phi_1 \in C_0 \setminus \mathcal{F}(\Sigma, \mathcal{V})$ is selected in the *adaption step* and some schematic variable renaming ν_1 and some schematic substitution π_1 is computed such that $\{\nu_0, \pi_0\} \cup \{\nu_1, \pi_1\}$ is admissible for $C \cup \{\Phi\}$ and $\nu_1 \circ \pi_1(\Phi_1) \in \mathcal{F}(\Sigma, \mathcal{V})$. The composed substitution $\nu_1 \circ \pi_1$ is called a *partial adaption candidate* of C_0 and is applied to C_0 yielding $C_1 := \nu_1 \circ \pi_1(C_0)$. If $C_1 \not\subset \mathcal{F}(\Sigma, \mathcal{V})$, further partial adaption candidates $\nu_2 \circ \pi_2$, $\nu_3 \circ \pi_3$ etc. are computed for C_1, C_2 etc. until eventually some totally instantiated catch $C_n := \nu_n \circ \pi_n(\ldots \nu_0 \circ \pi_0(C) \ldots) \subset \mathcal{F}(\Sigma, \mathcal{V})$ is obtained. Then $\nu_n \circ \pi_n \circ \cdots \circ \nu_1 \circ \pi_1$ is a *total adaption candidate* for $\nu_0 \circ \pi_0(C)$.

Finally, the system is recursively called in the *verification step* for proving $AX \models \psi'$ for each $\psi' \in C_n$. If successful, $\nu_n \circ \pi_n \circ \cdots \circ \nu_1 \circ \pi_1$ is an *adaption matcher* for $\nu_0 \circ \pi_0(C)$ and $AX \models \psi$ is verified by reuse. The recursive calls of the system necessitate *recursive calls of reuse* for the members of C_n, and each ψ' for which reuse fails must be proved *directly* by the theorem prover. In such a case the proof dictionary is extended by a further proof shell based on the analyzed proof of ψ'. Also the original conjecture ψ must be proved *directly* by the theorem prover with a subsequent extension of the proof dictionary, if the

attempt to prove ψ by reuse fails, i.e., the verification of $AX \models \psi'$ fails for some $\psi' \in C_n$ or no proof shells can be retrieved for ψ. [14]

7.1. Second-order matching

Second-order matching is the key concept for the computation of retrieval and adaption matchers. Since second-order matching is decidable and finitary, cf. [41], we can imagine an algorithm $match(p, t)$ which solves each solvable matching problem $[p \lhd t]$, i.e., computes a finite set $\{\mu_1 \circ \rho_1, \ldots, \mu_n \circ \rho_n\}$ of second-order substitutions for a *pattern* $p \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$ and a *target* $t \in \mathcal{T}(\Sigma, \mathcal{V})$ such that $\mu_i \circ \rho_i(p) = t$ for all $i \in \{1, \ldots, n\}$, where μ_i is a schematic variable renaming and ρ_i is a schematic substitution. We may apply *match* also to patterns $p \in \mathcal{F}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$ and targets $t \in \mathcal{F}(\Sigma, \mathcal{V})$ by treating quantifiers and connectives like function symbols.

To incorporate sortal reasoning, the matching algorithm is supplied with further arguments Π and M which hold the *context* under which the matchers are computed. Π is a finite set of schematic variable renamings or schematic substitutions and M is a finite set of mixed formulas, and we demand in addition that $\{\mu_i, \rho_i\} \cup \Pi$ is admissible for M for each $\mu_i \circ \rho_i \in match(p, t, \Pi, M)$. See Appendix A for a definition of the matching procedure incorporating sorts.

7.2. Retrieval of proof shells

Retrieval is the task of selecting a proof shell *PS* from a proof dictionary and computing substitutions π, ν for a given conjecture ψ such that *PS applies* for ψ via π, ν:

Definition 7.1 (*Proof shell applies*). A proof shell $PS = \langle \Phi, C \rangle$ *applies for* a conjecture $\psi \in \mathcal{F}(\Sigma, \mathcal{V})$ via a schematic substitution $\pi : \Omega \to \mathcal{T}(\Sigma, \mathcal{W})$ and a first-order variable renaming $\nu : \mathcal{U} \to \mathcal{V}$ iff

(1) $\nu \circ \pi(\Phi) = \psi$, and

(2) $\{\pi, \nu\}$ is admissible for $C \cup \{\Phi\}$.

Here requirement (1) states that $\nu \circ \pi$ has to be a (syntactical) schematic matcher for Φ and ψ, where π is a pure and closed substitution for function variables, cf. Definition 4.2, and ν renames first-order variables. Requirement (1) can be relaxed to $\nu \circ \pi(\Phi) \simeq \psi$ where $\simeq$ allows several equivalence preserving transformations like swapping the sides of equations, reordering subformulas or skolemizing universally quantified variables, etc. These extensions to syntactical matching can either be included in the matching algorithm, cf. [19], or be dealt with in a preprocessing step, cf. [50].

Applicability of a proof shell $\langle \Phi, C \rangle$ for a conjecture ψ is decided by calling $match(\Phi, \psi, \emptyset, C \cup \{\Phi\})$ and all applicable proof shells are recognized by considering each member of the proof dictionary.

[14] If the *simplifier* component of an *induction* prover (cf. Section 1) shall be supported by reuse, $C_n \subset Th_{ind}(AX)$ is tested instead of $AX \models C_n$, which may necessitate recursive calls of reuse for the *induction formulas* computed for some members of C_n.

However, this selection process has several indeterminisms which have to be resolved by suitable heuristics:

(i) *One* proof shell can apply for a conjecture via *several* matchers.
(ii) *Several* proof shells with *the same* schematic conjecture can apply for a conjecture.
(iii) *Several* proof shells with *different* schematic conjectures can apply for a conjecture.

Indeterminism (i) is based on the non-uniqueness of second-order matching, where, e.g., the matching problem $[F(G(x),x) \lhd \mathsf{f}(\mathsf{g}(x),x)]$ has the solutions

$$\pi_1 := \{F/\mathsf{f}(w_1,w_2),\ G/\mathsf{g}(w_1)\}, \qquad \pi_2 := \{F/\mathsf{f}(\mathsf{g}(w_1),w_2),\ G/w_1\},$$
$$\pi_3 := \{F/\mathsf{f}(\mathsf{g}(w_1),w_1),\ G/w_1\}, \qquad \pi_4 := \{F/\mathsf{f}(\mathsf{g}(w_2),w_1),\ G/w_1\},$$
$$\pi_5 := \{F/\mathsf{f}(\mathsf{g}(w_2),w_2),\ G/w_1\}, \qquad \pi_6 := \{F/w_1,\ G/\mathsf{f}(\mathsf{g}(w_1),w_1)\}.$$

To resolve this indeterminism, we propose a heuristic for rating and selecting matchers according to their *structural complexity*. For example, π_1 is considered as the less complex matcher since it is close to replacing function variables by function symbols only, while the other solutions are considered as more complex. The underlying motivation is to select a matcher which preserves the given structure of the proof being reused as far as possible, because it is more likely to find a valid instance of the schematic catch in this case. For rating matchers heuristically, we use a hierarchy $\mathcal{C}_n^i \subseteq \mathcal{T}(\Sigma,\mathcal{W}_n)$ of functional terms whose complexity increases with i (based on term structure and our experience on occurrences of functional terms in matchers from examples):

- $\mathcal{C}_n^0 := \{f(w_1,\ldots,w_n) \mid f \in \Sigma_n\}$: Function variables are replaced by function symbols of same arity while respecting the order of arguments.
- $\mathcal{C}_n^1 := \{f(w_{i_1},\ldots,w_{i_n}) \mid f \in \Sigma_n,\ \{i_1,\ldots,i_n\} = \{1,\ldots,n\}\}$: Function variables are replaced by function symbols of same arity with possible rearrangement of the argument order.
- $\mathcal{C}_n^2 := \{f(w_{i_1},\ldots,w_{i_m}) \mid f \in \Sigma_m,\ i_1,\ldots,i_m \in \{1,\ldots,n\}\}$: Function variables are replaced by function symbols. Here the arguments of function symbols can be rearranged, multiplied, or omitted.
- $\mathcal{C}_n^3 := \mathcal{W}_n \cup \mathcal{C}_n^2$: This set additionally includes all projections (including identity).
- $\mathcal{C}_n^4 := \{f_1(f_2(\ldots f_k(w_1,\ldots,w_n)\ldots)) \mid f_1,\ldots,f_{k-1} \in \Sigma_1,\ f_k \in \Sigma_n,\ k \geqslant 1\}$: This set includes compositions of (several) unary function symbols with one n-ary function.
- $\mathcal{C}_n^5 := \{f_1(f_2(\ldots f_k(w_{i_1},\ldots,w_{i_m})\ldots)) \mid f_1,\ldots,f_{k-1} \in \Sigma_1,\ f_k \in \Sigma_m,\ k \geqslant 1,\ i_1,\ldots,i_m \in \{1,\ldots,n\}\}$: This set combines the features of $\mathcal{C}_n^2$ and $\mathcal{C}_n^4$.
- $\mathcal{C}_n^6 := \mathcal{T}'(\Sigma,\mathcal{W}_n)$ which is recursively defined by
  - $\mathcal{W}_n \subseteq \mathcal{T}'(\Sigma,\mathcal{W}_n)$,
  - $f(w_{i_1},\ldots,w_{i_{j-1}},t,w_{i_{j+1}},\ldots,w_{i_k}) \in \mathcal{T}'(\Sigma,\mathcal{W}_n)$ if $f \in \Sigma_k$, $t \in \mathcal{T}'(\Sigma,\mathcal{W}_n)$, and $i_1,\ldots,i_{j-1},i_{j+1},\ldots,i_k \leqslant n$:

  This set includes all combinations of projections, compositions and argument reorderings where (recursively) at most one argument position of a functional term may be occupied by a non-variable term.
- $\mathcal{C}_n^7 := \mathcal{T}(\Sigma,\mathcal{W}_n)$.

For a fixed n, we find

$$\mathcal{C}_n^0 \subseteq \mathcal{C}_n^1 \subseteq \mathcal{C}_n^2 \subseteq \mathcal{C}_n^3, \quad \mathcal{C}_n^4 \subseteq \mathcal{C}_n^5 \subseteq \mathcal{C}_n^6, \quad \mathcal{C}_n^0 \subseteq \mathcal{C}_n^4, \quad \mathcal{C}_n^2 \subseteq \mathcal{C}_n^5, \quad \mathcal{C}_n^3 \subseteq \mathcal{C}_n^6,$$

i.e., the classes form a hierarchy of decreasing simplicity. For a matcher π, we let

$$rate(\pi) := \sum_{F \in \mathrm{dom}(\pi)} rate(\min\{i \mid F \in \Omega_n,\ \pi(F) \in \mathcal{C}_n^i\},\ \pi(F))$$

be a measure for the complexity of the replacing functional terms, where $rate(i,t)$ is a heuristically determined function yielding a natural number for rating the complexity of a functional term $t \in \mathcal{T}(\Sigma, \mathcal{W})$. We use

$$rate(i,t) := i + \#_\Sigma(t)$$

where $\#_\Sigma : \mathcal{T}(\Sigma, \mathcal{W}) \to \mathbb{N}$ yields the number of occurrences of function symbols in a (functional) term since this definition has proved useful in many experiments. Now indeterminisms of kind (i) are resolved by selecting one matcher π with minimal $rate(\pi)$ among all retrieval matchers. For obtaining an efficient implementation, the selection of the matcher is built into the matching algorithm such that only the selectable matcher is computed.

Our approach for reusing proofs is based on the heuristical assumption that *similar conjectures* have *similar proofs*. However, this heuristic must fail for certain statements and so it may happen that proof shells share the *same* schematic *conjecture* (up to a renaming of the symbols in the schematic conjectures), but have *different* schematic *catches* which results in indeterminisms of kind (ii). For instance, consider the proof shells

$$PS_1 = \langle\, F(A, G(v,w)) \equiv H(K(B,v), w),\\ \{F(A,u) \equiv u,\ K(B,u) \equiv u,\ G(u,u') \equiv H(u,u')\}\rangle,$$

and

$$PS_2 = \langle\, F(A, G(v,w)) \equiv H(K(B,v), w),\\ \{F(A,u) \equiv A',\ K(B,u) \equiv B',\ H(B',u) \equiv A')\}\rangle$$

which stem, e.g., from the analyzed proofs of the base case for the associativity of plus and times respectively. Both proof shells apply exactly for the same conjectures, like, e.g., $\mathsf{min}(0, \mathsf{min}(y,z)) \equiv \mathsf{min}(\mathsf{min}(0,y), z)$ and $\mathsf{max}(0, \mathsf{max}(y,z)) \equiv \mathsf{max}(\mathsf{max}(0,y), z)$, and a heuristic selection between the applicable proof shells is required.

To this effect, we compare the (partially instantiated) catches of each applicable proof shell by estimating the expected effort for computing an adaptation matcher. We prefer catches with a minimal number of function variables, a minimal number of proof obligations, and a maximal ratio of previous reuse successes versus failures. However, this choice point gives also room for backtracking, human guidance, or more sophisticated selection criteria.

The retrieval of proof shells is supported, if *all* proof shells $\langle \Phi, C_1 \rangle, \ldots, \langle \Phi, C_n \rangle$ sharing the same conjecture Φ are grouped into a *proof volume* $PV = \langle \Phi, \{C_1, \ldots, C_n\} \rangle$, and the proof dictionary is organized as a set of proof volumes instead. A proof volume represents *different* "proof ideas" for *one* statement, and retrieval is supported since a proof dictionary based on proof *volumes* has less schematic conjectures to be considered than a proof dictionary based on proof *shells*, cf. [54,57].

Indeterminism (iii) results from the flexibility of second-order matching because, e.g., the structurally different patterns $p_1 := F(G(x), y)$ and $p_2 := H(x, z, K(y))$ both match the target $t := a(y, b(x))$ via the matchers $\pi_1 := \{F/\mathsf{a}(w_2, w_1), G/\mathsf{b}(w_1)\}$ and $\pi_2 := \{H/\mathsf{a}(w_3, \mathsf{b}(w_1)),\ K/w_1\}$. While we could also use the heuristic developed for (i) to select the applicable proof volume with the least complex retrieval matcher (if high reusability is emphasized for the price of less efficient retrieval), we propose a further restriction of the class of useful matchers:

A matcher $\pi : \Omega \to \mathcal{T}(\Sigma, \mathcal{W})$ is called *simple* iff $F \in \mathrm{dom}(\pi) \cap \Omega_n$ implies $\pi(F) = f(w_{\alpha(1)}, \ldots, w_{\alpha(n)})$ for some $f \in \Sigma_n$ and some permutation α of $\{1, \ldots, n\}$. Hence a simple matcher replaces function variables by function symbols of the same arity where however the order of arguments may be rearanged, thus covering a fairly large class of useful matchers. A proof volume *simply applies* (*s-applies*) for a conjecture ψ iff it applies for ψ via a simple matcher, and a proof dictionary is called *s-minimal* iff for each conjecture ψ at most one proof volume s-applies for ψ. S-minimal proof dictionaries are obtained if the proof shells which form the proof volumes are based on the results of the *refined* analysis, cf. [54].

S-minimal proof dictionaries and s-application reduces the number of retrieval matchers considerably, however for the price reduced reusability. If more than one retrieval matcher is computed for a given conjecture, the final selection is based on the heuristic developed for case (i).

7.3. Adaptation of proof shells

After selection of a proof shell $PS = \langle \Phi, C \rangle$ from the proof dictionary and the computation of a retrieval matcher $\nu_0 \circ \pi_0$, some mixed formula $\Phi_1 \notin \mathcal{F}(\Sigma, \mathcal{V})$ is selected from the partially instantiated catch $\nu_0 \circ \pi_0(C)$, and a partial adaption candidate $\nu_1 \circ \pi_1$ must be computed for Φ_1. Since we aim to compute an adaption matcher, Φ_1 must be matched with some object formula $\varphi \in \mathcal{F}(\Sigma, \mathcal{V})$ satisfying $AX \models \varphi$, which means that matching is performed under the *theory* defined by the set of axioms AX. However, the problem of deciding whether an adaption *candidate* also is an adaption *matcher* is undecidable because semantical entailment is. Consequently heuristics are required also for the adaption step, which however strongly depend on the domain under consideration represented by the axioms AX. Therefore it seems useful to develop adaption heuristics from heuristics developed for *automated* theorem proving in the domain under consideration, such that in most cases the computed adaption *candidates* are in fact adaption *matchers*.

In the induction theorem proving domain, for instance, *defining equations* are applied to terms in an induction formula guided by the heuristic of *symbolic evaluation*, cf. [82]: In this domain each function symbol either is a *constructor* of some data structure or a *defined function symbol* specified by some axioms. For instance the declaration

structure empty add(number, list) : list

defines a data structure for linear lists of natural numbers with the constructors empty and add, and the axioms (len-1), (len-2) from Example 6.16 define the function len computing the length of a list. A term t is *evaluated* by applying the defining equations

for the function symbols contained in t. Thus, e.g., $\mathsf{len}(\mathsf{add}(\mathsf{len}(\mathsf{empty}), z))$ is evaluated via $\mathsf{len}(\mathsf{add}(0, z))$ to $\mathsf{s}(\mathsf{len}(z))$ by applying the defining equations for len. This is called *symbolic* evaluation because the term to be evaluated may contain variables as, e.g., z (which are not evaluated). Note that only defining equations are considered for the evaluation, i.e., the term $\mathsf{s}(\mathsf{plus}(\mathsf{plus}(x, y), z))$ remains non-evaluable, even if, e.g., the associativity of plus is given as a lemma.

Here we assume a *symbolic evaluator* which is implemented by a *terminating* operation *eval* and computes the normal form of a term such that

$$(47) \quad eval(l) \in \mathcal{T}(\Sigma, \mathcal{V}), \quad EQ \models_S l \equiv eval(l), \quad eval(eval(l)) = eval(l)$$

holds for all $l \in \mathcal{T}(\Sigma, \mathcal{V})$ and a set $AX = EQ \cup L$ of axioms.

We use the symbolic evaluator for the computation of adaption candidates in the following way: In a first step, some mixed equation $l_1 \equiv R_1$ (or $R_1 \equiv l_1$) is selected from the partially instantiated catch $C_1 := \nu_0 \circ \pi_0(C)$ such that $l_1 \in \mathcal{T}(\Sigma, \mathcal{V})$ and $R_1 \notin \mathcal{T}(\Sigma, \mathcal{V})$. We give up, if no such equation exists and we compute $\Pi_1 := match(R_1, eval(l_1), \{\nu_0, \pi_0\}, C \cup \{\Phi\})$ otherwise. If $\Pi_1 \neq \emptyset$, i.e., matching is successful, a partial adaption candidate $\nu_1 \circ \pi_1$ is selected from Π_1. Otherwise we define $\nu_1 := \pi_1 := \varepsilon$ and $l_1 \equiv R_1$ is inserted into a so-called remainder set X. Then we continue with $C_2 := \nu_1 \circ \pi_1(C_1 \backslash \{l_1 \equiv R_1\})$, i.e., some equation $l_2 \equiv R_2$ is selected from C_2 such that $l_2 \in \mathcal{T}(\Sigma, \mathcal{V})$ and $R_2 \notin \mathcal{T}(\Sigma, \mathcal{V})$. We give up, if no such equation exists in $C_2 \neq \emptyset$ and we compute $\Pi_2 := match(R_2, eval(l_2), \{\nu_1, \pi_1\, \nu_0, \pi_0\}, C \cup \{\Phi\})$ otherwise, etc.

Adaption always terminates with some set C_n which contains no mixed equation $l \equiv R$ (or $R \equiv l$) such that $l \in \mathcal{T}(\Sigma, \mathcal{V})$ and $R \notin \mathcal{T}(\Sigma, \mathcal{V})$. Adaption has failed, if

$$C^* := C_n \cup \nu_n \circ \pi_n(\ldots \nu_1 \circ \pi_1(X) \ldots) \not\subseteq \mathcal{F}(\Sigma, \mathcal{V}).$$

Otherwise $\nu_n \circ \pi_n \circ \cdots \circ \nu_1 \circ \pi_1$ is the computed total adaption candidate for $\nu_0 \circ \pi_0(C)$ and C^* is the totally instantiated catch (with some already proved equations removed).

We call this principle for computing adaption candidates *matching modulo evaluation*, because it combines automated reasoning with syntactical second-order matching, as we attempt to solve the matching problem $[R \lhd eval(l)]$ instead of $[R \lhd l]$.

The procedure for computing adaption candidates has two choice points which require further determination:

(1) The selection of a mixed equation $l_i \equiv R_i$ from C_i is guided by a heuristic which prefers the mixed equation which has probably the least number of matchers ρ solving $eval(l_i) = \rho(R_i)$, i.e., the most "constrained" of the equations (without function variables on the lhs) is chosen. We consider an equation with a high number of function symbols and a low number of function variables as "highly constrained", because both criteria limit the number of possible matchers, cf. [55].

(2) The selection of an adaption candidate $\nu_i \circ \pi_i$ from Π_i is guided by a heuristic which prefers the "simplest" matcher solving the chosen equation $l \equiv R$, i.e., the number of function symbols introduced by ρ is minimal, cf. [55]. For example, $\{F/w_2\}$ is preferred to $\{F/\mathsf{len}(w_1)\}$ for the equation $\mathsf{len}(x) \equiv F(x, \mathsf{len}(x))$. The underlying motivation is to select a matcher which preserves the given structure of the proof being reused, because it is more likely to find a valid instance of the schematic proof in this case.

Example 7.2 (*Matching modulo evaluation*). Reconsider conjecture $\psi_4 := \mathsf{plus}(\mathsf{len}(x), \mathsf{len}(y)) \equiv \mathsf{len}(\mathsf{app}(x, y))$ and its step formula ψ_{4s} from Example 6.16. Since the proof shell $PS_1 = \langle \Phi, C \rangle$ from Fig. 4 applies for ψ_{4s} via the matcher $\pi := \{F^1/\mathsf{plus}, G^{1,2,3}/\mathsf{len}, H^1/\mathsf{app}, D^1/\mathsf{add}\}$ and an appropriate first-order variable renaming ν, cf. Example 6.16, the partially instantiated catch

$$C_p := \pi(C) = \left\{ \begin{array}{ll} (48) & \mathsf{len}(\mathsf{add}(n, x)) \equiv F^2(n, \mathsf{len}(x)) \\ (49) & \mathsf{app}(\mathsf{add}(n, x), y) \equiv D^4(n, \mathsf{app}(x, y)) \\ (50) & \mathsf{len}(D^4(n, x)) \equiv F^3(n, \mathsf{len}(x)) \\ (51) & \mathsf{plus}(F^2(x, y), z) \equiv F^3(x, \mathsf{plus}(y, z)) \end{array} \right\}$$

is computed for the step case.

We choose Eq. (48) and *evaluate* the left hand side (lhs) of (48) by the defining axiom (len-2) yielding $\mathsf{s}(\mathsf{len}(x)) \equiv F^2(n, \mathsf{len}(x))$. Now the lhs matches the rhs via the unique matcher $\rho_1 := \{F^2/\mathsf{s}(w_2)\}$ which means that (48) is solved because $\rho_1((48)) = (\mathsf{len}\text{-}2)$. We apply ρ_1 to the remaining mixed formulas and obtain

$$\rho_1(C_p \setminus \{(48)\}) := \left\{ \begin{array}{ll} (49) & \mathsf{app}(\mathsf{add}(n, x), y) \equiv D^4(n, \mathsf{app}(x, y)) \\ (50) & \mathsf{len}(D^4(n, x)) \equiv F^3(n, \mathsf{len}(x)) \\ (52) & \mathsf{plus}(\mathsf{s}(y), z) \equiv F^3(x, \mathsf{plus}(y, z)) \end{array} \right\}.$$

We continue with the next mixed equation whose (say) lhs does not contain function variables, i.e., with (49) or with (52). Here we choose (49) and evaluating the lhs using a defining axiom for app yields $\mathsf{add}(n, \mathsf{app}(x, y))$ which matches the rhs via the unique matcher $\rho_2 := \{D^4/\mathsf{add}\}$. Now the lhs of $\rho_2((50))$ is first-order and can be evaluated, i.e., $\rho_3 := \{F^3/\mathsf{s}(w_2)\}$ is obtained by matching, and all free function variables are instantiated by the adaptation candidate $\rho := \{F^2/\mathsf{s}(w_2),\ F^3/\mathsf{s}(w_2),\ D^4/\mathsf{add}\}$. Since the remaining proof obligation $\rho((52)) = \mathsf{plus}(\mathsf{s}(y), z) \equiv \mathsf{s}(\mathsf{plus}(y, z))$ is a defining axiom for plus, ρ is an adaptation matcher and the proof reuse is completed successfully.

Note that symbolic evaluation is essential for the success of adaptation. For example, in the example above, no adaptation candidate can be computed if the *unevaluated* (object) terms of the equations are used as targets for the matching problems. See [55] for a more detailed account on *matching modulo evaluation*.

After a totally instantiated catch is computed by retrieval and adaption, the resulting proof obligations are subject to further verification which may necessitate *recursive calls* of reuse. While recursive reuse generally increases the potential savings of search costs, it also introduces the problem of *termination*. For instance, the partially instantiated catch $\{\mathsf{g}(\mathsf{b}) \equiv A'', \mathsf{f}(A'') \equiv \mathsf{a}\}$ is computed for proving $\mathsf{f}(\mathsf{g}(\mathsf{b})) \equiv \mathsf{a}$ by reuse based on the proof shell PS^* from Example 6.2, but the (simplest) solution $\{A''/\mathsf{g}(\mathsf{b})\}$ for the matching

problem $[A'' \lhd \mathsf{g(b)}]$ yields $\mathsf{f(g(b))} \equiv \mathsf{a}$ as the remaining proof obligation. Since this is exactly the conjecture under consideration, recursive reuse will never terminate in this example. A remedy to this problem is to impose restrictions on adaption candidates which avoids infinite reuse attempts. However, such restrictions also depend on the reasoning domain under consideration. See [56–58] for restrictions guaranteeing termination of recursive reuse in the domain of *induction theorem proving* (where all examples in this paper obey these restrictions).

8. Experiments with reusing proofs

We have implemented a prototype called the PLAGIATOR system [13,51] which consists of a device for analyzing, generalizing and managing proofs and is based on the techniques discussed above. If a statement cannot be verified by reuse, the user must support the system with a proof which then is analyzed, etc. The creation of a hand crafted proof is supported by a proof editor which is also part of the system. The proof editor only checks whether inference rules are legally applied but offers no further support in finding the proof. This means that the system is really weak in its problem solving ability, thus motivating the system's name—the German word for plagiarist—as it is intended to obtain a system exhibiting an intelligent behavior only because it is able to adapt solutions provided by other intelligent devices.

Thus apart from the initial proofs provided by a human advisor in the "prove" step, none of the steps of the implemented reuse procedure necessitates human support. Hence, e.g., the proof shell of Fig. 4 is automatically reused for proving the step formulas of the *apparently different* conjectures φ_i given in Table 1 below.

This table illustrates a typical session with the PLAGIATOR system: At the beginning of the session, the human advisor submits statement φ_0 (in the first row) and a proof p of φ_0 to the system. Then the statements $\varphi_1, \varphi_2, \ldots$ are presented to the PLAGIATOR, which proves the step formula for each statement only by reuse of p such that no user interactions are required. The third column shows the subgoals speculated by the system when proving a statement by reuse, i.e., the proof obligations which are returned after instiating the schematic catch. Here "$-$" denotes that all proof obligations are simplified to tautologies by evaluation (i.e., the statement is proved by reuse only), and "[...]" denotes that heuristics different from the heuristics given in Section 7 are used. For example, statement φ_{17} is speculated when verifying φ_{16}, which leads to speculating φ_{18} which in turn entails speculation of conjecture φ_{19}, for which eventually φ_{25} is speculated. For φ_5 an *instance* of conjecture φ_{11} is speculated, viz. the formula $\sigma_5(\varphi_{11})$ with $\sigma_5 = \{p/m :: \varepsilon\}$. The speculated conjectures $\varphi_{53}, \ldots, \varphi_{56}$ cannot be proved by reusing the proof of φ_0:

$$\varphi_{53} := \mathsf{max}(m, \mathsf{max}(n, i)) \equiv \mathsf{max}(\mathsf{max}(m, n), i).$$

$$\varphi_{54} := \mathsf{min}(m, \mathsf{min}(n, i)) \equiv \mathsf{min}(\mathsf{min}(m, n), i).$$

$$\varphi_{55} := \mathsf{or}(\mathsf{or}(\mathsf{eq}(m, n), a), b) \equiv \mathsf{or}(\mathsf{eq}(m, n), \mathsf{or}(a, b)).$$

$$\varphi_{56} := \mathsf{if}(\mathsf{eq}(m, n), k, n :: l) <> p \equiv \mathsf{if}(\mathsf{eq}(m, n), k <> p, n :: (l <> p)).$$

Table 1
Conjectures proved and lemmata speculated by reuse of φ_0 [15]

φ_0	$\sum k + \sum l \equiv \sum(k <> l)$	φ_{25}
Φ	$F^1(G^1(x), G^2(y)) \equiv G^3(H^1(x, y))$	

No.	Conjectures proved by reuse	Subgoals
φ_1	$\prod k \times \prod l \equiv \prod(k <> l)$	φ_{18}
φ_2	$\mid k <> l \mid \equiv \mid l <> k \mid$	φ_{12}
φ_3	$\mid \mathsf{rev}(k) \mid \equiv \mid k \mid$	φ_{13}
φ_4	$\mathsf{rev}(\mathsf{rev}(k)) \equiv k$	φ_{14}
φ_5	$\mathsf{rev}(l) <> \mathsf{rev}(k) \equiv \mathsf{rev}(k <> l)$	$\sigma_5(\varphi_{11})$
φ_6	$\mathsf{max}(\mathsf{maxl}(k), \mathsf{maxl}(l)) \equiv \mathsf{maxl}(k <> l)$	φ_{53}
φ_7	$\mathsf{min}(\mathsf{minl}(\mathsf{n}, k), \mathsf{minl}(\mathsf{n}, l)) \equiv \mathsf{minl}(\mathsf{n}, k <> l)$	φ_{54}
φ_8	$\mathsf{plusl}(\mathsf{m}, k) <> \mathsf{plusl}(\mathsf{m}, l) \equiv \mathsf{plusl}(\mathsf{m}, k <> l)$	–
φ_9	$\mid k \mid + \mid l \mid \equiv \mid k <> l \mid$	–
φ_{10}	$\mathsf{ncut}(m, \mathsf{ncut}(n, \mathsf{k})) \equiv \mathsf{ncut}(\mathsf{plus}(m, n), \mathsf{k})$	–
φ_{11}	$k <> (l <> \mathsf{p}) \equiv (k <> l) <> \mathsf{p}$	–
φ_{12}	$\mid k <> n :: \mathsf{l} \mid \equiv \mathsf{s}(\mid k <> \mathsf{l} \mid)$	–
φ_{13}	$\mid k <> n :: \varepsilon \mid \equiv \mathsf{s}(\mid k \mid)$	–
φ_{14}	$\mathsf{rev}(k <> n :: \varepsilon) \equiv n :: \mathsf{rev}(k)$	–
φ_{15}	$k <> \varepsilon \equiv k$	–
φ_{16}	$(\mathsf{i}^{\mathsf{n}})^m \equiv \mathsf{i}^{m \times \mathsf{n}}$	$[\varphi_{17}]$
φ_{17}	$\mathsf{i}^m \times \mathsf{i}^n \equiv \mathsf{i}^{m+n}$	φ_{18}
φ_{18}	$m \times (\mathsf{n} \times \mathsf{i}) \equiv (m \times \mathsf{n}) \times \mathsf{i}$	$[\varphi_{19}]$
φ_{19}	$m \times \mathsf{i} + n \times \mathsf{i} \equiv (m + n) \times \mathsf{i}$	φ_{25}
φ_{20}	$m \times \mathsf{i} + n \times \mathsf{i} \equiv \mathsf{i} \times (m + n)$	$[\varphi_{22}, \varphi_{25}]$
φ_{21}	$m \times \mathsf{n} \equiv \mathsf{n} \times m$	$[\varphi_{22}]$
φ_{22}	$m \times \mathsf{s}(\mathsf{n}) \equiv m + m \times \mathsf{n}$	$[\varphi_{23}]$
φ_{23}	$m + (\mathsf{i} + n) \equiv \mathsf{i} + (m + n)$	φ_{26}
φ_{24}	$m + n \equiv n + m$	φ_{26}
φ_{25}	$m + (n + \mathsf{i}) \equiv (m + n) + \mathsf{i}$	–
φ_{26}	$m + \mathsf{s}(n) \equiv \mathsf{s}(m + n)$	–
φ_{27}	$m + 0 \equiv m$	–
φ_{28}	$m \times 0 \equiv 0$	$[-]$
φ_{29}	$m \times \mathsf{s}(0) \equiv m$	$[-]$
φ_{30}	$2m + 2n \equiv 2(m + n)$	–
φ_{31}	$\mathsf{or}(\mathsf{mem}(\mathsf{m}, k), \mathsf{mem}(\mathsf{m}, l)) \equiv \mathsf{mem}(\mathsf{m}, k <> l)$	φ_{55}
φ_{32}	$\mathsf{rm}(\mathsf{m}, k) <> \mathsf{rm}(\mathsf{m}, l) \equiv \mathsf{rm}(\mathsf{m}, k <> l)$	φ_{56}

If we let π_i denote the matcher for Φ and the step formula of φ_i, then, e.g., the matcher $\pi_1 = \{F^1/\times,\ D^1/\mathsf{add}, \ldots\}$ replaces only function variables with function symbols, whereas, e.g., π_{30} with $\pi_{30}(D^1) = \mathsf{s}(w_2)$ makes use of functional terms. This allows to reuse the proof of the step formula of φ_0 involving the data structure list for a conjecture concerning the data structure number. Similarly $\pi_5(F^1) = \mathsf{app}(w_2, w_1)$ swaps the arguments of F^1 and $\pi_{26}(G^1) = w_1$ uses a projection to match the schematic step formula. The applicability of a proof shell can be increased if we "freeze variables to constants", i.e., a (universally quantified) variable $z \in \mathcal{V}$ is regarded as a new constant $\mathsf{z} \in \Sigma_0$. Freezing variables allows to match a function variable $F \in \Omega_n$ with an object term having *more* than n variables. Thus, e.g., conjecture ψ_{19}, i.e., the distributivity law for $\times$ and $+$, matches the schematic conjecture Φ because now $\pi_{19}(G^{1,2,3}) = \times(w_1, \mathsf{i}) \in \mathcal{T}(\Sigma, \mathcal{W})$ can be used for the matcher.[16] In the cases denoted by "[...]" the retrieval heuristic from Section 7 chooses an unsuccessful matcher. For an alternative successful retrieval matcher provided by the user, however, the adaptation heuristic usually works also in these cases.

9. Related work

The way of proving theorems by mathematical induction as used here is called *explicit induction* due to the explicit computation and application of induction axioms. Explicit induction has been implemented successfully in several induction theorem proving systems like NQTHM [11,12], INKA [5,43], and OYSTER-CLAM [15,17], which are (sometimes partly) based on the following techniques:

(i) computation of induction axioms [80,81] from well-founded orderings, which themselves are obtained from termination proofs for algorithms [35,83],
(ii) guiding the proof of induction formulas by *rippling* [16,42],
(iii) generalization and lemma speculation [44,45,82].

In [70] a method for combining (i) and (ii) by postponing the commitment to a fixed induction axiom is presented. When applying our reuse method to induction theorem proving, we assume that induction axioms (i) are computed elsewhere (since we reuse only the first-order part of a proof), but we provide an alternative technique for (ii) and (iii).

Note that most of the proofs of the statements in our examples can be obtained by *rippling* [16,42]. This method attempts to reduce syntactical differences between an induction conclusion and the induction hypotheses by application of lemmata and defining equations in a goal directed way. To this effect, equations carry annotations denoting

[15] For the sake of readability we use mathematical (infix) symbols for functions where appropriate, i.e., $\times$, $+$, $-$, $<>$, $|\,.\,|$, $::$, $\sum$, and $\prod$ denote times, plus, minus, app, len, add, sum, and prod, respectively. Further rev reverses a list, max respectively maxl (min respectively minl) computes the maximum (minimum) of two numbers respectively a list of numbers, plusl sums two lists of numbers pairwise, ncut removes the last elements of a list, mem decides list membership, and rm removes all occurrences of an element of a list. We use the convention with respect to variable names that i, j, n, m denote numbers, k, l, p, q denote lists, and x, y, z are variables of any sort.

[16] This kind of skolemization is sound because $\forall z.\psi[z]$ is valid iff $\psi[\mathsf{z}]$ is valid for a Skolem constant z and a formula $\psi[z]$ with a free variable z.

the syntactical difference which is reduced when the equation is applied. Since rippling and our reuse proposal are based on different principles, one method may fail while the other succeeds for a certain proof problem. For example, our proposal is not restricted to the induction domain, whereas rippling requires an induction conclusion and induction hypotheses to compute the differences which must be reduced. On the other hand, a statement which is easily proved by rippling cannot be proved by reuse, if no appropriate proof shell is available. Rippling also fails if the given equations cannot be annotated. For example, the step formulas ψ_{5s} and ψ_{6s} from Example 6.16(iii) cannot be proved by rippling, because (by the mutual recursion) no annotation for each of the defining equations (dhalf-2), (uhalf-2), (even-2) and (odd-2) can be computed. But step formula ψ_{6s} can be easily proved by reuse as the required adaption matcher is easily computed by *matching modulo evaluation*.

The utilization of past problem solving experience has attracted researchers right from the beginning of AI and several (considerably different) methodologies have been proposed during the years. We briefly sketch those of these proposals which can be applied to theorem proving, and discuss similarities and significant differences with our method.

9.1. Reasoning by analogy

Analogical reasoning (AR) uses a representational mapping to map problems and solutions from a *source* into a *target* domain, cf. [38,39,48] and Fig. 8(iv). If a new problem (in the target domain) can be represented as the image of a solved source problem under the representational mapping (*recognition*), it is also tried to map the source problem's solution into the target domain (*elaboration*), and the result serves as a *candidate* for the target problem's solution, but has to be verified or repaired subsequently (*evaluation*). In present research of applying analogical reasoning to theorem proving, e.g. [9,10,14,68], a suitable source conjecture is given (or selected) and matched with the target conjecture, and then the source proof is mapped to the target domain step by step, *guiding* the target proof. Thus elaboration and evaluation are interleaved, where *repairing* the proposed target proof (e.g., by adding or removing intermediate inference steps) is necessary if a step suggested by the analogy cannot be applied. This leads to new problems concerning the control of the repairing actions.

Viewed in our framework, the representational mapping is initially given by the second-order matching between conjectures and schematic conjectures (retrieval $\cong$ *recognition*), then extended by instantiating the free function variables from the schematic catch (adaptation $\cong$ *elaboration*) and subsequent verification of the resulting equations (prove $\cong$ *evaluation*). Hence in our approach the reuse is based on the *axioms* (used in the proof of a source problem) which have to be verified for the target domain, whereas the whole new *proof* can always be constructed in a uniform way by "patching" an instantiated schematic proof [53]. [17] The fact that our method for reusing proofs only considers the conjecture and the axioms used in the source problem's solution constitutes the main difference to other applications of analogical reasoning to theorem proving in which the reuse is based on the *proof structure* which has to be modified for increasing the reuse success. Consequently

[17] This is not required if one is concerned with *provability only*, i.e., the *existence* of some proof.

analogical reasoning demands that the whole proof is *replayed* to guarantee the soundness of the analogical inferences.

Our method provides an *increased flexibility* of reuses by the second-order substitutions such that no repair or replay is required, cf. Theorem 4.3. These benefits also hold for a further development of reasoning by analogy in a resolution logic, which is proposed in [20]. Here a given refutation of a clause set is generalized by replacing first-order terms with second-order terms and subsequent application of so-called *generalization rules*, which, e.g., separate different occurrences of function and predicate symbols (like in our proposal of *refined analysis*), instantiate variables, eliminate variable and function symbols, change the arity of function and predicate symbols, etc. These rules aim to generalize a refutation as much as possible but to avoid generalizations which necessitate an expensive computation of retrieval matchers or cause subsequent proofs by analogy to fail. If a new clause set S is to be investigated for unsatisfiability, S is matched with the generalizations already computed, and a refutation for S is obtained by applying a second-order matcher, for which the match succeeded, to the generalization. For instance, given a refutation of

$$S = \big\{\{\neg q(x_1, x_2), \neg p(x_1, x_3), p(x_2, x_3)\}, \{q(a, b)\}, \{p(a, a), p(b, b)\}, \{\neg p(b, x_4)\}\big\},$$

a refuation of

$$S_1 = \big\{\{\neg q(a), \neg p(b), r(c, d)\}, \{q(a)\}, \{p(b), r(e, d)\}, \{\neg r(x, d)\}\big\}$$

is obtained by the proposed analogy reasoning method. However the approach fails for the clause set

$$S_2 = \big\{\{\neg q(x_1, x_2), \neg p(x_1, x_3), r(x_2, x_3)\}, \{q(a, b)\}, \{p(a, c), s(d, e)\}, \{\neg s(d, x_4)\}\{\neg r(b, x_5)\}\big\}$$

as the generalization of the refutation of S does not match S_2. If we apply our reuse approach to resolution, the schematic catch of Fig. 7 is obtained from the refutation of S by refined analysis. Since clause sets are considered here, no schematic conjecture exists and a new clause set S_{new} must be matched with C_S to test for unsatisfiability. This test succeeds iff (*) $S_{new} \models_{sub} \pi(C_S)$ for some second-order substitution π, where $\models_{sub}$ denotes clause

$$C_S := \left\{ \begin{array}{ll} (1) & \{\neg Q(u_1, u_2),\ \neg P(u_1, u_3),\ R(u_2, u_3)\} \\ (2) & \{Q(A, B)\} \\ (3) & \{P(A, C),\ S(D, E)\} \\ (4) & \{\neg S(D, u_4)\} \\ (5) & \{\neg R(B, u_5)\} \end{array} \right\}$$

Fig. 7. The schematic catch C_S for the refutation of S.

set subsumption. This reasoning is sound, because C_S is unsatisfiable, hence $\pi(C_S)$ is unsatisfiable, and then S_{new} is unsatisfiable by (*). The second-order substitution

$$\pi := \{Q/q(a), P/p(b), R/r(c,d), S/r(w_2,w_1), D/d, E/e\}$$

satisfies requirement (*) for the clause set S_1, because $\pi(C_S) = S_1 \cup \{\{\neg r(c,d)\}\}$. Hence S_1 must be unsatisfiable, and obviously also the unsatisfiability of S_2 can be proved this way (because S_2 only is a one-to-one reformulation of C_S). But this increase of reusability comes with the prize of high search costs caused by the complexity of matching clause sets when computing a retrieval matcher π. Therefore techniques tailored for this domain, cf. [21], are more appropriate than our approach.

While we discussed the *transformational analogy* paradigm so far which is based on mapping problems and solutions, in the *derivational analogy* paradigm [6,18,76,77] rather the problem solving process is considered for this mapping. Hence this reuse technique is based on past problem solving experiences, cf. [61] for an application to theorem proving. The ABALONE system of Melis and Whittle [62] uses both transformational and derivational analogy in the domain of inductive theorem proving: Given a source theorem, a *proof plan* is computed whose execution yields the proof of the theorem. For a target theorem to be proved, the function symbols of the source and the axioms used in the proof are mapped to function symbols in the target and to conjectures which are required to prove the target. Then the proof plan is replayed step by step, and—if required—modified or reformulated in a goal directed way. Although there are similarities with our reuse proposal, (e.g., the association of the source and the target by second-order substitutions, indexing of function symbols to make different occurrences explicit, speculation of lemmata), both proposals differ significantly in two aspects:

(1) Whereas our reuse method is only concerned with first-order reasoning (i.e., the proofs of base and step cases), ABALONE covers also the computation of *induction schemas* and *generalizations*.
(2) Our reuse method is based on the analysis of a proof, whereas ABALONE's analogy method is based on a proof *plan*.

The latter feature in particular requires several rules for proof plan reformulation, which corresponds to the adaption step by *matching modulo evaluation* in our proposal. As all first-order examples in the paper can be easily solved by our reuse method, it cannot be concluded from the paper whether the increased effort for proof plan replay and reformulation also increases the success for analogy in fact. On the other hand, ABALONE fails for some of our examples, viz. φ_{17}, φ_{18} and φ_{19} from Table 1, as reported in [86].

9.2. Explanation-based learning

Our proposal can be viewed as a variant of *explanation-based learning* (EBL), cf. [22, 30,64,65,72,73] and also [28] for a survey. This paradigm aims to improve a problem solver by *generalizing* a concrete *problem solution* which is derived after explaining the solution of an example problem using given background knowledge: At first, a given problem is solved and the problem as well as its solution are generalized using the technique of *goal regression*, cf. Fig. 8(i). If a new problem is an instance of the generalized problem, the

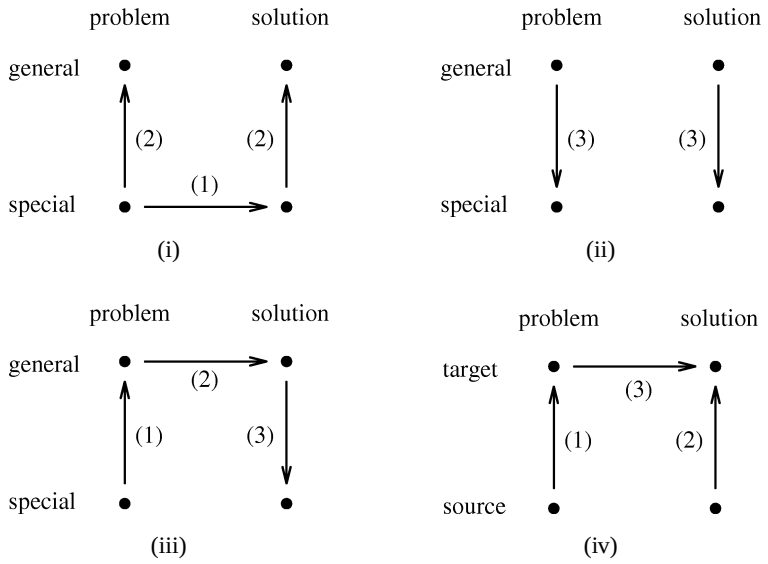

Fig. 8. Related machine learning paradigms; (i) Solve & analyse; (ii) Reuse; (iii) Abstraction; (iv) Analogy.

corresponding instance of the generalized solution is obtained and some specific features may remain which require subsequent solution steps, cf. Fig. 8(ii).

In EBL, the generalized problem is called a (*non-operational*) *goal concept* whereas the generalized solution is the corresponding *operational specialization*. The background knowledge is given by the *domain theory*. In our terminology a schematic conjecture respectively a schematic catch resembles a goal concept respectively its operational specialization and the domain theory is given by the initial axioms and lemmata. However, EBL merely provides a reformulation of a solution on the same level and lacks incorporation of abstractions in the sense of [36,63]. Applied to theorem proving, *goal regression* computes only EBL-generalizations which replace first-order terms by first-order variables, and this is far to weak for a frequent reuse, see [27,47,67,75]. We therefore replace first-order terms by (certain) second-order terms and this counts for a remarkable increase of reusability compared to the standard EBL approach, cf., e.g., Section 8 for examples.

The advantages of higher-order concepts were recognized in particular by Donat and Wallen [27] and by Dietzen and Pfenning [26]. Donat and Wallen generalized standard EBL using *higher-order resolution* in the domain of symbolic integration. In this domain, rules for symbolic integration are represented by (conditional) *production rules*, as, e.g.,

(1) $\int X^A \,\mathrm{d}(X) \mapsto (X^{A+1}/(A+1))$, if *constant*$(A)$ & $A \neq -1$,
(2) $\int C * A \,\mathrm{d}(X) \mapsto C * \int A \,\mathrm{d}(X)$, if *constant*$(C)$, and
(3) $\int sin(X) \,\mathrm{d}(X) \mapsto -cos(X)$.

These rules are applied to integration problems, as, e.g., $\int 3 * x^2 \,\mathrm{d}x$, yielding the *solution sequence*

(4) $\int 3 * x^2 \,\mathrm{d}x \mapsto 3 * \int x^2 \,\mathrm{d}x \mapsto 3 * (x^3/3)$.

From such a solution sequence a *generalized rule* then is computed as a higher-order resolvent of the production rules used in the sequence, and for the example

(5) $F(\int C * A\,\mathrm{d}(X)) \mapsto G(Z^{D+1}/(D+1))$, if *constant*$(D)$ & $D \neq -1$ & *constant*(C) & $F(C * \int A\,\mathrm{d}(Y)) = G(\int Z^D\,\mathrm{d}(Z))$ is obtained.

Generalized rules are applied to new integration problems like the given rules, and, e.g., $2 * (x^4/4)$ is is obtained by applying (5) to the integration problem $\int 2 * x^3\,\mathrm{d}x$. However, since a generalized rule encodes all rules used in a particular solution sequence, its application fails for integration problems which need to reorder applied rules or to replace them by other rules, cf. [27]. In particular, function symbols like $*$, *sin* and the symbol for exponentiation are not generalized and, for instance, only $3 * \int sin(x)\,\mathrm{d}x$ is computed, if (5) is applied to $\int 3 * sin(x)\,\mathrm{d}x$. The application of the generalized rule (5) fails, since rule (3) must be used instead of (1) to solve the integral. Applying our reuse approach to this domain, the proof shell of Fig. 9 is obtained from (4) by simple analysis.

Since $\int F(C, G(X, A))\,\mathrm{d}(X)$ matches $\int 3 * sin(x)\,\mathrm{d}x$, proof shell PS_{int} applies for $\int 3 * sin(x)\,\mathrm{d}x$ via the matcher $\pi := \{F/w_1 * w_2,\ C/3,\ G/sin(w_1),\ X/x\}$ yielding the partially instantiated proof shell of Fig. 10.

The partially instantiated catch of Fig. 10 is *solvable* (yielding after a simplification step $-3 * cos(x)$ as the result of the integration problem), as $\pi((1))$ is a variant of rule (2)

$$\Phi_{int} := \int F(C, G(X, A))\,\mathrm{d}(X) \mapsto F(C, H(G(X, A+1), A+1)) \quad \text{if } constant(C) \,\&\, constant(A) \,\&\, A \neq -1$$

$$C_{int} := \left\{ \begin{array}{ll} (1) & \int F(D, Y)\,\mathrm{d}(Z) \mapsto F(D, \int Y\,\mathrm{d}(Z)), \\ & \text{if } constant(D) \\ (2) & \int G(U, B)\,\mathrm{d}(U) \mapsto H(G(U, B+1), B+1), \\ & \text{if } constant(B) \,\&\, B \neq -1 \end{array} \right\}$$

Fig. 9. The proof shell PS_{int} for the solution sequence (4).

$$\pi(\Phi_{int}) := \int 3 * sin(x)\,\mathrm{d}x \mapsto 3 * H(sin(x), A+1) \quad \text{if } constant(A) \,\&\, A \neq -1$$

$$\pi(C_{int}) := \left\{ \begin{array}{ll} \pi((1)) & \int D * Y\,\mathrm{d}(Z) \mapsto D * \int Y\,\mathrm{d}(Z), \\ & \text{if } constant(D) \\ \pi((2)) & \int sin(U)\,\mathrm{d}(U) \mapsto H(sin(U), B+1), \\ & \text{if } constant(B) \,\&\, B \neq -1 \end{array} \right\}$$

Fig. 10. The partially instantiated proof shell $\pi(PS_{int})$.

and $\{H/-(1-w_1^2)^{1/2}\}$ solves $\pi((2))$, because $\int sin(U)\,\mathrm{d}(U) \mapsto -cos(U)$ by rule (3) and $cos(U) = (1-sin^2(U))^{1/2}$. A similar observation can be made for the proposal of Dietzen and Pfenning [26]. For example, the generalized rule

(6) $\int C * X^A\,\mathrm{d}x \mapsto C * (X^{A+1}/(A+1))$

is computed from the solution of $\int 3 * x^2\,\mathrm{d}x$, and also here $\int 3 * sin(x)\,\mathrm{d}x$ cannot be solved by (6). Our reuse proposal is independent of specific function symbols in the conjecture and of the order of rule applications, as the catch is a set whose members may be considered in any order. Therefore other rules may be involved in the adaption step, which counts for the success of our reuse proposal in the example above.

9.3. Abstraction techniques

Abstraction techniques distinguish a *basic* and an *abstract* level of problems and their solutions, cf. [4,36,49] and also [69,78] where this methodology is applied to theorem proving. A basic problem is

(1) mapped to an abstract problem by an *abstraction mapping*, then
(2) an abstract solution is computed for the abstract problem and finally
(3) a solution is obtained for the given problem by applying the inverted abstraction mapping to the abstract solution, cf. Fig. 8(iii).

Our method differs from that because we solve basic problems and generalize their solutions instead of solving generalized problems. The reuse step (3) is similar in both approaches, cf. Fig. 8(i) and (ii). The disadvantage of abstraction in the above sense is that (depending on the used abstract level) problem solving on the abstract level either is much more difficult than on the basic level as most control information is lost (e.g., consider our abstract level of formulas with second-order variables) or it is simple enough but the basic solution obtained in the reuse step (3) is incomplete, i.e., there are still large gaps that have to be closed subsequently by expensive basic problem solving. Therefore we use our abstract level rather for representational purposes as it provides rigorous and adaptable criteria for the relevant concepts, viz. similarity of conjectures, generalization, and instantiation.

9.4. Case-based reasoning

The rapidly growing field of *case-based reasoning* (CBR) [1,59,60,66,85] develops approaches which store whole solutions in a data base and rely either purely on (effective retrieval in) large case bases and sometimes also on adaptation techniques, e.g., [7,32, 74]. The main difference to our method is that cases in CBR only consist of a set of attribute-value pairs (sometimes enriched with information on the problem solving process). Therefore CBR does not seem to be very useful in knowledge-intensive domains like theorem proving, but has more success in domains where statistical methods based on similarity criteria (e.g., nearest neighbor concepts) apply. The PRODIGY system uses derivational analogy as a means to integrate general problem solving with CBR [77]. Here derivational analogy exploit past cases to solve similar problems so that a problem solver can successfully create its own case library. The case base consists of problems $P = (S, G)$, where S and G are sets of literals (formed with predicate and constant

symbols) representing the *initial state* S and the *goal state* G of P. For a new problem $P' = (S', G')$, the case base is searched for a similar problem (S, G), where (S, G) is similar to (S', G'), if some of the literals in S' "match" some members of S and some of the goals in G' "match" some members of G. Here "matching" means that constant symbols in P' are replaced by constant symbols of P so that identical literals are obtained after the replacement. The quality of "matching" is rated by the number of initial state literals and goals which "match" literals in S and G respectively. The *foot-print similarity metric* emphasizes goal oriented behaviour: By application of EBL techniques to the solution derivation of a problem (S, G), each member g of G is associated with those members of S which are necessary for achieving the goal g. "Matching" of the initial states now is constrained to those initial state literals from S, which are associated with the "matched" goals g.

9.5. Learning and theorem proving

Some approaches for learning in theorem proving have been developed which are based on the *learning of heuristics* for problem solving. For instance, [23,25,33,34] merge the experiences gained from several proofs by learning a specialized heuristic (respectively an instance of a parameterized heuristic) for the considered domain using genetic algorithms [33], adaptation of weights [34], or recognition of useful facts [23].

These methods are based on a property of the applied proof calculus (equational reasoning by unfailing Knuth–Bendix completion [3]), viz. that there is a central decision instance (for the selection of critical pairs) which is controlled by a heuristic. This also allows for an easy combination with from-scratch theorem proving, but makes it harder to combine several specialized heuristics for different purposes (for which the teamwork approach [2] is a partial remedy).

The method of [71] for reusing proofs in software verification is based on the replay of inference rules, where—after a failed proof attempt for a faulty conjecture and a subsequent modification of some axioms—parts of the failed proof are reused for a new proof attempt. A similar approach for generating tactics from proofs in an interactive environment is taken in [31].

10. Summary and conclusion

We have presented an approach for verifying theorems by reusing previously computed proofs: A proof is analyzed and the essentials of the proof are represented in a certain data structure, viz. a proof shell. For a new verification task, a suitable proof shell is selected in a (heuristically guided) retrieval step and subsequently adapted (under heuristic guidance) for the verification problem under consideration. This yields new "simpler" proof obligations which are subject to further verification, either *directly* by a *theorem prover* or *recursively* by *reuse* again.

This paper defines a formal framework for proof analysis, generalization, retrieval and adaptation, discusses the problems which arise, proposes solutions and verifies the soundness of the proposals. Our approach offers two benefits, as several experiments with

an implementation of a learning component for a theorem prover, viz. the PLAGIATOR system [13,51], reveal, cf. Section 8 and [57]:

(1) User interactions are saved, and
(2) the PLAGIATOR system is able to *speculate lemmata* by recognizing previously computed solutions, which are helpful to prove a given conjecture.

The latter feature is particularly important, because induction theorem proving is *incomplete* and therefore the speculation of useful lemmata may yield a relevant improvement of an induction prover's performance cf. [44,56,58].

Although these benefits can be observed in the "toy domain" of *unconditional* equational (inductive) theorem proving, it is not known to the authors how the method behaves in more ambitious domains, and, of course, an answer to this question is required to assess the general usefulness of the proposal. We believe that the postulated benefits also show up in more practical domains, but nevertheless it may also happen in the worst case that the problem of determining the *search* for a *proof* is simply turned into the problem of determining the *search* for *retrieval* and *adaptation* matchers, and then nothing is saved (and worse, costs are increased by the additional overhead). First investigations are encouraging [50], but a complete answer to this question is subject to further research to be carried out in the next stage of our project.

Acknowledgements

We like to thank D. Hutter and T. Walsh for clarifying rippling and the anonymous referees for constructive suggestions.

Appendix A. An algorithm for well-sorted second-order matching

Combining the tests (1) and (2) of Definition 7.1 demands for a *sorted* second-order matching algorithm, where in particular a (with respect to sort considerations) *partial* match has to be performed, since we must respect also the sort constraints imposed by C while matching only Φ. This is because we want to exclude as early as possible syntactical matchers for Φ and ψ for which there is no *well-sorted* total instantiation of C.

To treat a *formula*-pair $\langle \Phi, \psi \rangle$ with an algorithm *match* for (pairs of) *terms* to be developed subsequently from the algorithm of Huet and Lang [41], we let $\langle R, \nu \rangle := \mathit{decompose}(\Phi, \psi)$ denote the preprocessing step of structurally comparing Φ and ψ (up to quantified variables and terms in equations). For example, $R := [F(u) \lhd a(x),\ G(v) \lhd b,\ H(u,v) \lhd f(y),\ D \lhd c]$ and $\nu := \{u/x,\ v/y\}$ results from decomposing $\forall u \forall v\ F(u) \equiv G(v) \wedge H(u,v) \equiv D$ and $\forall x \forall y\ a(x) \equiv b \wedge f(y) \equiv c$.[18] Hence ν is a renaming and R contains pairs $p \lhd t$ of pattern terms $p \in \mathcal{T}(\Omega, \mathcal{U})$ and target terms $t \in \mathcal{T}(\Sigma, \mathcal{V})$ if *decompose* succeeds, while $R = [\,]$ otherwise denotes that Φ and ψ are structurally different.

[18] We use a PROLOG-style list notation $[a_1, \ldots, a_n]$ respectively $[a|R]$.

Thus a proof shell $PS = \langle \Phi, C \rangle$ applies for a conjecture ψ via π and ν iff $\langle R, \nu \rangle := \mathit{decompose}(\Phi, \psi)$ succeeds, π is a matcher for the *matching problem* $\nu(R) := [p_1 \lhd t_1, \ldots, p_n \lhd t_n]$, i.e., $\pi(p_i) = t_i$ for all $i \in \{1, \ldots, n\}$, and $\{\pi, \nu\}$ is admissible for $C \cup \{\Phi\}$. We present a matching calculus for computing π which is invoked with the triple $\langle \nu(R), \{\nu\}, \{\} \rangle$ and operates with respect to a fixed set of mixed formulas $M := C \cup \{\Phi\}$:

Definition A.1 (*Many-sorted second-order matching calculus*). The *many-sorted second-order matching calculus* consists of four rules operating on triples $\langle R, \Pi, \pi \rangle$ where the input component $R := [p_1 \lhd t_1, \ldots, p_n \lhd t_n]$ is a matching problem with respect to a fixed set of mixed formulas M, the input component Π is a set of schematic substitutions and variable renamings, and the output component π is a schematic substitution. Let $x \in \mathcal{V}$, $f \in \Sigma_n$, $p_1, \ldots, p_n \in \mathcal{T}(\Sigma \cup \Omega, \mathcal{V} \cup \mathcal{U})$, $t_1, \ldots, t_n, \ldots, t_m, t \in \mathcal{T}(\Sigma, \mathcal{V})$, $F \in \Omega_n$, and $g \in \Sigma_m$:

- *Variable*
$$\frac{\langle [x \lhd x \mid R], \Pi, \pi \rangle}{\langle R, \Pi, \pi \rangle}.$$
- *Decomposition*
$$\frac{\langle [f(p_1, \ldots, p_n) \lhd f(t_1, \ldots, t_n) \mid R], \Pi, \pi \rangle}{\langle [p_1 \lhd t_1, \ldots, p_n \lhd t_n \mid R], \Pi, \pi \rangle}.$$
- *Projection*
$$\frac{\langle [F(p_1, \ldots, p_n) \lhd t \mid R], \Pi, \pi \rangle}{\langle [\pi_i(p_i) \lhd t \mid \pi_i(R)], \Pi_i, \pi_i \circ \pi \rangle}$$
if $\Pi_i := \Pi \cup \{\pi_i\}$ is admissible for M, where $\pi_i := \{F/w_i\}$ for some $i \in \{1, \ldots, n\}$.
- *Imitation*
$$\frac{\langle [F(p_1, \ldots, p_n) \lhd g(t_1, \ldots, t_m) \mid R], \Pi, \pi \rangle}{\langle [G_1(\pi_0(p_1), \ldots, \pi_0(p_n)) \lhd t_1, \ldots, G_m(\pi_0(p_1), \ldots, \pi_0(p_n)) \lhd t_m \mid \pi_0(R)], \Pi_0, \ \pi_0 \circ \pi \rangle}$$
if $\Pi_0 := \Pi \cup \{\pi_0\}$ is admissible for M, where $\pi_0 := \{F/g(G_1(w_1, \ldots, w_n), \ldots, G_m(w_1, \ldots, w_n))\}$ for new [19] function variables $G_1, \ldots, G_m \in \Omega_n$.

A sequence $\langle \langle R_1, \Pi_1, \pi_1 \rangle, \ldots, \langle R_n, \Pi_n, \pi_n \rangle \rangle$ of triples of matching problems R_i, sets of schematic substitutions and renamings Π_i, and schematic substitutions π_i is a *derivation* in the sorted second-order matching calculus with respect to a fixed set of mixed formulas M if for each $i \in \{1, \ldots, n-1\}$, $\langle R_{i+1}, \Pi_{i+1}, \pi_{i+1} \rangle$ results from applying one of the rules to $\langle R_i, \Pi_i, \pi_i \rangle$.

A schematic substitution π' is called a *solution* for a matching problem R with respect to Π and M iff a derivation $\langle \langle R, \Pi, \{\} \rangle, \ldots, \langle [\,], \Pi', \pi' \rangle \rangle$ exists, and we let

$$\mathit{match}(R, \Pi, M) := \big\{ \pi \mid \pi := \{F/t \in \pi' \mid F \in \Omega(R)\},\ \pi' \text{ is a solution for } R \text{ with respect to } \Pi \text{ and } M \big\}$$

denote the set of all solutions for R with respect to Π and M where the domains of the obtained substitutions are limited to the symbols occurring in R (excluding introduced function variables).

[19] Here *new* means that in particular Π and M do not contain any of the function variables G_j.

Theorem A.2 (Matching of proof shells). *Let* $R := [p_1 \lhd t_1, \ldots, p_n \lhd t_n]$ *be a matching problem with respect to a fixed set of mixed formulas* M *and let* Π *be a set of schematic substitutions and variable renamings, which is admissible for* M.

(i) (Soundness) *If* $\langle\langle R, \Pi, \emptyset\rangle, \ldots, \langle[], \Pi', \pi'\rangle\rangle$ *is a derivation in the sorted matching calculus then* $\pi'(p_i) = t_i$ *for all* $p_i \lhd t_i \in R$ *and* $\Pi \cup \pi'$ *is admissible for* M.

(ii) (Completness modulo most Generality) *If* $\pi''(p_i) = t_i$ *for all* $p_i \lhd t_i \in R$ *such that* $\Pi \cup \pi''$ *is admissible for* M, *then there is a derivation* $\langle\langle R, \Pi, \emptyset\rangle, \ldots, \langle[], \Pi', \pi'\rangle\rangle$ *in the sorted matching calculus for some* $\pi' \subseteq \pi''$.

Proof. Follows from [41] (respectively [40]) since we have only adapted the procedure of *Huet and Lang* to our notation and our restrictions. Admissibility is guaranteed by the tests in the rules of the calculus, because Π' as the result of the derivation is admissible for M and π' is composed from members of Π'. □

Since the matching calculus is *locally finite* (i.e., each configuration only has finitely many successors) and also each derivation is finite, a terminating, sound and complete matching algorithm is obtained by the derivation of *all* solutions for a sorted matching problem.

References

[1] A. Aamodt, E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches, AI Communications 7 (1) (1994) 39–59.

[2] J. Avenhaus, J. Denzinger, M. Fuchs, DISCOUNT: A system for distributed equational deduction, in: Proc. 6th International Conference on Rewriting Techniques and Applications (RTA-95), Kaiserslautern, Germany, Lecture Notes in Computer Science, Vol. 914, Springer, Berlin, 1995, pp. 397–402.

[3] L. Bachmair, N. Dershowitz, D.A. Plaisted, Completion without failure, in: Colloquium on the Resolution of Equations in Algebraic Structures, Austin (1987), Academic Press, New York, 1989.

[4] R. Bergmann, W. Wilke, Learning abstract planning cases, in: N. Lavrac, S. Wrobel (Eds.), Machine Learning: ECML-95 (Proc. European Conference on Machine Learning, 1995), Heraklion, Greece, Lecture Notes in Artificial Intelligence, Vol. 914, Springer, Berlin, 1995, pp. 55–76.

[5] S. Biundo, B. Hummel, D. Hutter, C. Walther, The Karlsruhe induction theorem proving system, in: Proc. 8th International Conference on Automated Deduction (CADE-86), Oxford, UK, 1986, pp. 672–674.

[6] B. Blumenthal, B.W. Porter, Analysis and empirical studies of derivational analogy, Artificial Intelligence 67 (1994) 287–327.

[7] K. Börner, C.H. Coulon, E. Pippig, E.-C. Tammer, Structural similarity and adaption, in: I. Smith, B. Faltings (Eds.), Proc. 3rd European Workshop on Case-Based Reasoning (EWCBR-96), Springer, Berlin, 1996, pp. 58–75.

[8] A. Bouhoula, E. Kounalis, M. Rusinowitch, SPIKE: An automatic theorem prover, in: Proc. Conference on Logic Programming and Automated Reasoning (LPAR-92), St. Petersburg, Russia, Springer, Berlin, 1992.

[9] T. Boy de la Tour, R. Caferra, Proof analogy in interactive theorem proving: A method to express and use it via second order pattern matching, in: Proc. AAAI-87, Seattle, WA, 1987, pp. 95–99.

[10] T. Boy de la Tour, C. Kreitz, Building proofs by analogy via the Curry–Howard isomorphism, in: Proc. Conference on Logic Programming and Automated Reasoning (LPAR-92), St. Petersburg, Russia, Springer, Berlin, 1992, pp. 202–213.

[11] R.S. Boyer, J.S. Moore, A Computational Logic, ACM Monograph Series, Academic Press, New York, 1979.

[12] R.S. Boyer, J.S. Moore, A Computational Logic Handbook, Perspectives in Computing, Vol. 23, Academic Press, New York, 1988.

[13] J. Brauburger, PLAGIATOR—Design and implementation of a learning theorem prover, Diploma Thesis (in German), TH Darmstadt, 1994.
[14] B. Brock, S. Cooper, W. Pierce, Analogical reasoning and proof discovery, in: Proc. 9th International Conference on Automated Deduction (CADE-88), Argonne, IL, 1988, pp. 454–468.
[15] A. Bundy, The use of explicit plans to guide inductive proofs, in: Proc. 9th International Conference on Automated Deduction (CADE-88), Argonne, IL, Springer, Berlin, 1988, pp. 111–120.
[16] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, A. Smaill, Rippling: A heuristic for guiding inductive proofs, Artificial Intelligence 62 (1993) 183–253.
[17] A. Bundy, F. van Harmelen, C. Horn, A. Smaill, The Oyster–Clam system, in: Proc. 10th International Conference on Automated Deduction (CADE-90), Kaiserslautern, Germany, 1990, pp. 647–648.
[18] J.G. Carbonell, Derivational analogy: A theory of reconstructive problem solving and expertise acquisition, in: R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 2, Morgan Kaufmann, San Mateo, CA, 1986, Chapter 14, pp. 371–392.
[19] R. Curien, Second order E-matching as a tool for automated theorem proving, in: Progress in Artificial Intelligence (Proccedings EPIA '93), 1993, pp. 242–257.
[20] G. Defourneaux, C. Bourely, N. Peltier, Semantic generalizations for proving and disproving conjectures by analogy, J. Automat. Reason. 20 (1998) 27–45.
[21] G. Defourneaux, N. Peltier, Partial matching for analogy discovery in proofs and counter-examples, in: Proc. 14th International Conference on Automated Deduction (CADE-97), Townsville, Australia, 1997.
[22] G. DeJong, R. Mooney, Explanation-based learning: An alternative view, Machine Learning 1 (1986) 145–176.
[23] J. Denzinger, S. Schulz, Learning domain knowledge to improve theorem proving, in: M. McRobbie, J. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ, Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 62–76.
[24] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science: Formal Models and Semantics, Vol. B, Elsevier Science, Amsterdam, 1990, Chapter 6, pp. 243–320.
[25] R.V. Desimone, Learning Control Knowledge within an Explanation-Based Learning Framework, Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, 1989.
[26] S. Dietzen, F. Pfenning, Higher-order and modal logic as a framework for explanation-based generalization, Machine Learning 9 (1992) 23–55.
[27] M.R. Donat, L.A. Wallen, Learning and applying generalised solutions using higher order resolution, in: Proc. 9th International Conference on Automated Deduction (CADE-88), Argonne, IL, 1988, pp. 41–60.
[28] T. Ellman, Explanation-based learning: A survey of programs and perspectives, ACM Computing Surveys 21 (2) (1989) 163–221.
[29] H.B. Enderton, A Mathematical Introduction to Logic, Academic Press, San Diego, CA, 1972.
[30] O. Etzioni, A structural theory of explanation-based learning, Artificial Intelligence 60 (1993) 93–139.
[31] A. Felty, D. Howe, Generalization and reuse of tactic proofs, in: F. Pfenning (Ed.), Proc. 5th Conference on Logic Programming and Automated Reasoning (LPAR-94), Lecture Notes in Artificial Intelligence, Vol. 822, Springer, Berlin, 1994, pp. 1–15.
[32] A.G. Francis, A. Ram, A domain-independent algorithm for multi-plan adaptation and merging in least-commitment planners, in: D.W. Aha, A. Ram (Eds.), Adaptation of Knowledge for Reuse. Papers from the 1995 AAAI Fall Symposium, Cambridge, MA, AAAI Press, 1995, pp. 19–25.
[33] M. Fuchs, Learning proof heuristics by adapting parameters, in: Proc. 12th International Conference on Machine Learning, Tahoe City, CA, 1995.
[34] M. Fuchs, Experiments in the heuristic use of past proof experience, in: M. McRobbie, J. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ, Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 523–537.
[35] J. Giesl, Termination analysis for functional programs using term orderings, in: Proc. 2nd International Static Analysis Symposium (SAS-95), Glasgow, Scotland, Lecture Notes in Computer Science, Vol. 983, Springer, Berlin, 1995, pp. 154–171.
[36] F. Giunchiglia, T. Walsh, A theory of abstraction, Artificial Intelligence 57 (1992) 323–389.
[37] W.D. Goldfarb, The undecidability of the second-order unification problem, Theoret. Comput. Sci. 13 (1981) 225–230.

[38] R. Greiner, Learning by understanding analogies, Artificial Intelligence 35 (1988) 81–125.
[39] R.P. Hall, Computational approaches to analogical reasoning: A comparative analysis, Artificial Intelligence 39 (1989) 39–120.
[40] G. Huet, A unification algorithm for typed λ-calculus, Theoret. Comput. Sci. 1 (1975) 27–57.
[41] G. Huet, B. Lang, Proving and applying program transformations expressed with second-order patterns, Acta Informatica 11 (1978) 11–31.
[42] D. Hutter, Colouring terms to control equational reasoning, J. Automat. Reason. 18 (1997) 399–442.
[43] D. Hutter, C. Sengler, INKA: The next generation, in: M. McRobbie, J. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ, Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 288–292.
[44] A. Ireland, A. Bundy, Productive use of failure in inductive proof, J. Automat. Reason. (Special Issue on Automation of Proofs by Mathematical Induction) 16 (1–2) (1996).
[45] D. Kapur, M. Subramaniam, Lemma discovery in automating induction, in: M. McRobbie, J. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ, Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 538–552.
[46] D. Kapur, H. Zhang, RRL: A rewrite rule laboratory, in: E. Lusk, R. Overbeek (Eds.), Proc. 9th International Conference on Automated Deduction (CADE-88), Argonne, IL, Lecture Notes in Computer Science, Vol. 310, Springer, Berlin, 1988, pp. 768–769.
[47] S.T. Kedar-Cabelli, L.T. McCarty, Explanation-based generalization as resolution theorem proving, in: Proc. 4th International Workshop on Machine Learning, Irvine, CA, 1987, pp. 93–106.
[48] R.E. Kling, A paradigm for reasoning by analogy, Artificial Intelligence 2 (1971) 147–178.
[49] C.A. Knoblock, Automatically generating abstractions for planning, Artificial Intelligence 68 (1994) 243–302.
[50] T. Kolbe, Optimizing proof search by machine learning techniques, Doctoral Dissertation, Technische Hochschule Darmstadt, Shaker, Aachen, 1997.
[51] T. Kolbe, J. Brauburger, PLAGIATOR—A learning prover, in: W. McCune (Ed.), Proc. 14th International Conference on Automated Deduction (CADE-97), Townsville, Australia, Lecture Notes in Artificial Intelligence, Vol. 1249, Springer, Berlin, 1997, pp. 256–259.
[52] T. Kolbe, S. Glesner, Many-sorted logic in a learning theorem prover, in: G. Brewka, C. Habel, B. Nebel (Eds.), Proc. 21st German Annual Conference on Artificial Intelligence (KI-97), Freiburg, Lecture Notes in Artificial Intelligence, Vol. 1303, Springer, Berlin, 1997, pp. 75–86.
[53] T. Kolbe, C. Walther, Patching proofs for reuse, in: N. Lavrac, S. Wrobel (Eds.), Proc. European Conference on Machine Learning (ECML-95), Heraklion, Greece, Lecture Notes in Artificial Intelligence, Vol. 912, Springer, Berlin, 1995, pp. 303–306.
[54] T. Kolbe, C. Walther, Proof management and retrieval, in: Proc. IJCAI'95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs, Montreal, Quebec, 1995, pp. 16–20.
[55] T. Kolbe, C. Walther, Second-order matching modulo evaluation—A technique for reusing proofs, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 190–195.
[56] T. Kolbe, C. Walther, Termination of theorem proving by reuse, in: M. McRobbie, J. Slaney (Eds.), Proc. 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ, Lecture Notes in Artificial Intelligence, Vol. 1104, Springer, Berlin, 1996, pp. 106–120.
[57] T. Kolbe, C. Walther, Proof analysis, generalization and reuse, in: W. Bibel, P.H. Schmitt (Eds.), Automated Deduction—A Basis for Applications, Vol. II, Systems and Implementation Techniques, Applied Logic Series, Vol. 9, Kluwer Academic, Dordrecht, 1998, pp. 189–219.
[58] T. Kolbe, C. Walther, On terminating lemma speculations, J. Inform. Comput. (1999), to appear.
[59] J.L. Kolodner, Case-Based Reasoning, Morgan Kaufmann, San Mateo, CA, 1993.
[60] R. Lopez de Mantaras, E. Plaza, Case-Based Reasoning, MLnet News, 1995.
[61] E. Melis, A model of analogy-driven proof-plan construction, in: Proc. IJCAI-95, Montreal, Quebec, Morgan Kaufmann, San Mateo, CA, 1995, pp. 182–188.
[62] E. Melis, J. Whittle, Analogy in inductive theorem proving, J. Automat. Reason. 22 (1999) 117–147.
[63] R.S. Michalski, Y. Kodratoff, Research in machine learning: Recent progress, Classification of methods and future directions, in: Y. Kodratoff, R.S. Michalski (Eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 3, Morgan Kaufmann, San Mateo, CA, 1990, pp. 3–30.

[64] S. Minton, Quantitative results concerning the utility of explanation-based learning, Artificial Intelligence 42 (1990) 363–391.
[65] T.M. Mitchell, R.M. Keller, S.T. Kedar-Cabelli, Explanation-based generalization: A unifying view, Machine Learning 1 (1986) 47–80.
[66] G. Nakhaeizadeh, N. Fuhr, K. Morik, B. Bartsch-Spörl, S. Wess, Zur Diskussion, Künstliche Intelligenz 1 (1996) 36–41. Themenheft Fallbasiertes Schließen.
[67] X. Nie, D.A. Plaisted, Application of explanation-based generalization in resolution theorem proving, in: Z. Ras (Ed.), Methodologies for Intelligent Systems 4, North-Holland, New York, 1989, pp. 226–233.
[68] S. Owen, Analogy for Automated Reasoning, Academic Press, New York, 1990.
[69] D.A. Plaisted, Theorem proving with abstraction, Artificial Intelligence 16 (1981) 47–108.
[70] M. Protzen, Lazy generation of induction hypotheses, in: Proc. 12th International Conference on Automated Deduction (CADE-94), Nancy, France, 1994.
[71] W. Reif, K. Stenzel, Reuse of proofs in software verification, in: R. Shyamasundar (Ed.), Foundation of Software Technology and Theoretical Computer Science, Bombay, India, 1993.
[72] S. Schrödl, Explanation-based generalization for negation as failure and multiple examples, in: Proc. 12th European Conference on Artificial Intelligence (ECAI-96), Budapest, Hungary, Wiley, New York, 1996, pp. 448–452.
[73] A. Segre, C. Elkan, A high-performance explanation-based learning algorithm, Artificial Intelligence 69 (1994) 1–50.
[74] B. Smyth, P. Cunningham, Deja Vu: A hierarchical case-based reasoning system for software design, in: Proc. 10th European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria, 1992.
[75] F. van Harmelen, A. Bundy, Explanation-based generalisation = partial evaluation, Artificial Intelligence 36 (1988) 401–412.
[76] M.M. Veloso, PRODIGY/ANALOGY: Analogical reasoning in general problem solving, in: S. Wess, K.-D. Althoff, M.M. Richter (Eds.), Topics in Case-Based Reasoning—Proceedings of the 1st European Workshop on Case-Based Reasoning, Kaiserslautern, Germany, Lecture Notes in Artificial Intelligence, Vol. 837, Springer, Berlin, 1993, pp. 33–50.
[77] M.M. Veloso, J.G. Carbonell, Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization, Machine Learning 10 (1993) 249–278.
[78] A. Villafiorita, F. Giunchiglia, Inductive theorem proving via abstraction, in: Proc. 4th International Symposium on Artificial Intelligence and Mathematics, 1996.
[79] C. Walther, A Many-Sorted Calculus Based on Resolution and Paramodulation, Research Notes on Artificial Intelligence, Pitman, London/Morgan Kaufmann, Los Altos, CA, 1987.
[80] C. Walther, Computing induction axioms, in: Proc. Conference on Logic Programming and Automated Reasoning (LPAR-92), St. Petersburg, Russia, 1992.
[81] C. Walther, Combining induction axioms by machine, in: Proc. IJCAI-93, Chambery, France, 1993.
[82] C. Walther, Mathematical induction, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 2, Oxford University Press, Oxford, 1994, pp. 127–227.
[83] C. Walther, On proving the termination of algorithms by machine, Artificial Intelligence 71 (1) (1994) 101–157.
[84] C. Walther, T. Kolbe, Report on proving theorems by reuse, Technical Report, Technische Universität Darmstadt, 1998.
[85] I.D. Watson, An introduction to case-based reasoning, in: I.D. Watson (Ed.), Progress in Case-Based Reasoning, Lecture Notes in Artificial Intelligence, Vol. 1020, Springer, Berlin, 1995, pp. 3–16.
[86] J. Whittle, Analogy in CLAM, Master's Thesis, Department of Artificial Intelligence, University of Edinburgh, 1995.